



A short introduction to the development and evaluation of Indexing systems

Danilo Croce

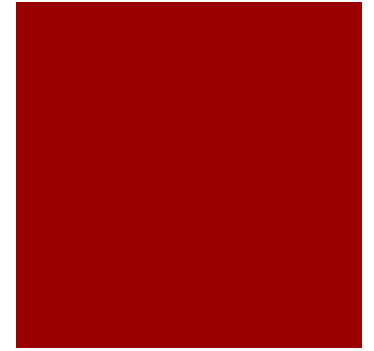
`croce@info.uniroma2.it`

Information Retrieval
2022-2023

Dec 2022

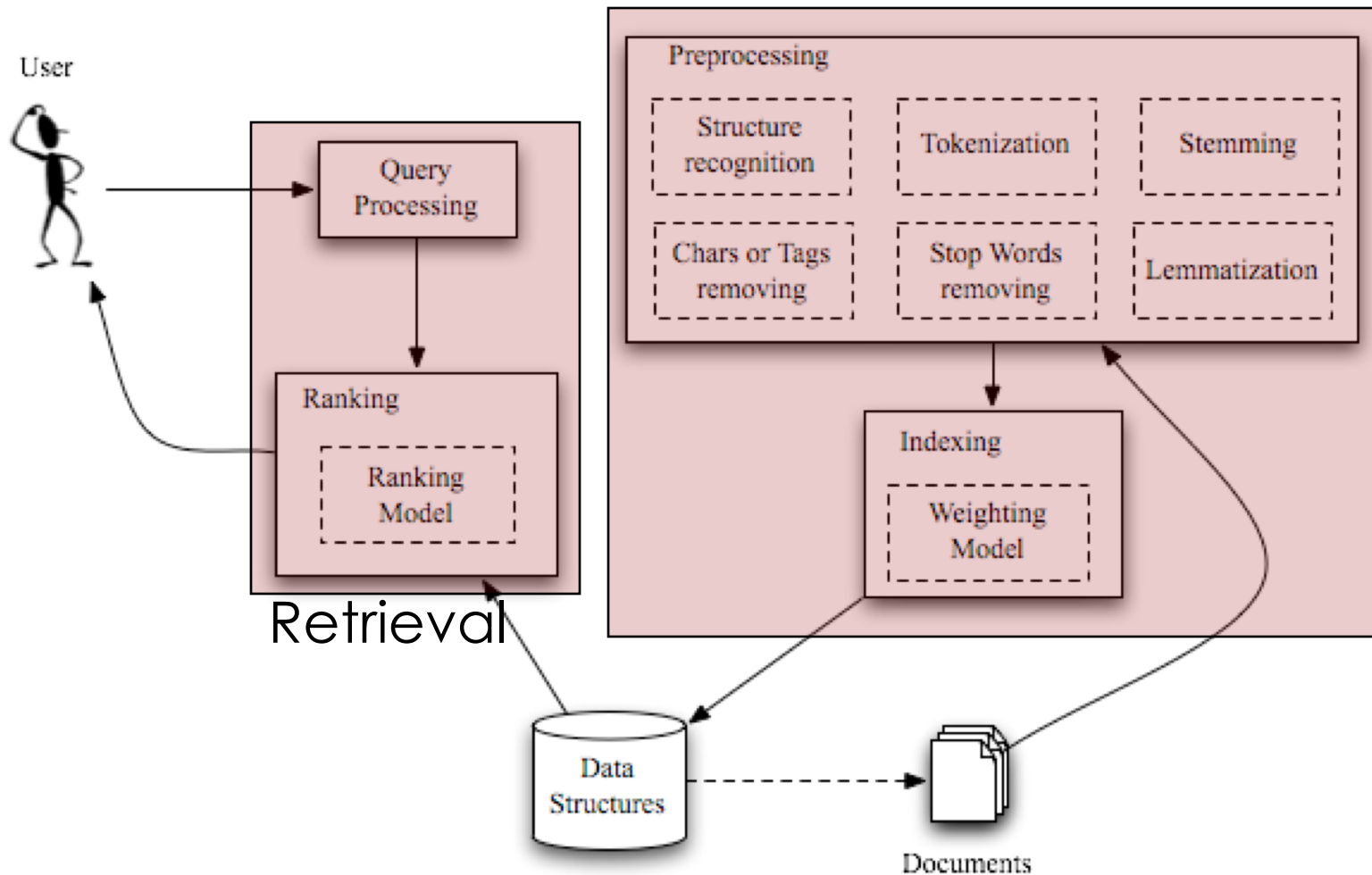
Outline

- An introduction to Lucene
 - Main features
- Implementation of a first Retrieval Engine
 - Luke: a graphical user interface to Lucene
- Exercises
 - Evaluation of a Retrieval Engine



Generic Architecture of a Information Retrieval Engine

Indexing



Lucene



- Lucene is a Search Engine library with many features including
 - fast indexing,
 - ranked searching,
 - boolean, phrase, and span queries,
 - date-range searching,
- Written in JAVA
- Lucene is open-source and released under the Apache License
- Objective: high scalability and customization
- <https://lucene.apache.org/>

Lucene – Main features



- Web pages can be indexed by adopting a structured view
 - Each document is composed of different **fields**
- Several pre-processing facilities have been implemented for different languages
- Several types of queries can be used to retrieve documents
 - phrase queries, wildcard queries, proximity queries, range queries, etc...
- Documents are retrieved by exploiting
 - Single fields, separately used
 - Multiple fields (results are grouped and a single rank is provided)

A general view of Lucene



- A Lucene index represents documents as it follows
 - An **index** contains a sequence of **documents**
 - Each **document** is made of a set of **fields**
 - A **field** is a sequence of **terms**
 - Each field **does not depend** from the other fields
 - The same string occurring in two different field represent two different terms

Index

Document

Document

Document

Document

Field

Field

Field

Term

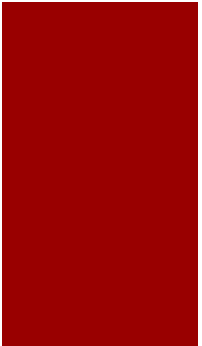
Term

Term

Term

Term

Term



Inverted Indexing



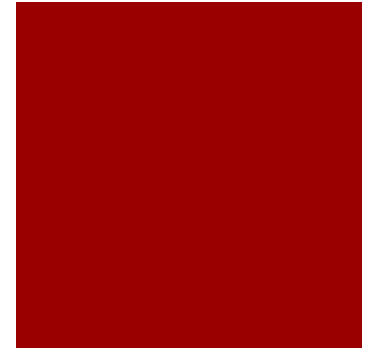
- The index stores statistics about terms in order to make term-based search more efficient.
- Lucene's index falls into the family of indexes known as an inverted index.
 - it can list, for a term, the documents that contain it.
 - This is the inverse of the natural relationship, in which documents list terms.
- Once a query is submitted to the system
 - Query terms (composing the query) are used to efficiently recover the documents containing them

Overview of a Lucene Index



- **Field names.** This contains the set of field names used in the index.
- **Stored Field values.** This contains, for each document, a list of attribute-value pairs, where the attributes are field names.
 - These are used to store auxiliary information about the document, such as its title, url, or an identifier to access a database.
 - The set of stored fields are what is returned for each hit when searching.
 - This is keyed by document number.
- **Term dictionary.** A dictionary containing all of the terms used in all of the indexed fields of all of the documents.
 - The dictionary also contains the number of documents which contain the term, and pointers to the term's frequency and proximity data.

Overview of a Lucene Index



- **Term Frequency data.** For each term in the dictionary, the numbers of all the documents that contain that term
- **Term Proximity data.** For each term in the dictionary, the positions that the term occurs in each document.
- **Normalization factors.** For each field in each document, a value is stored that is multiplied into the score for hits on that field.

Structure of the Lucene Project



- `org.apache.lucene.document`: it contains the classes used to represent a document
- `org.apache.lucene.analysis`: it contains the classes needed to pre-process documents
 - used to determine the set of terms
 - used both before indexing and search
 - the same pre-processor must be used for indexing and search
- `org.apache.lucene.index`: it contains the classes needed to represent an index

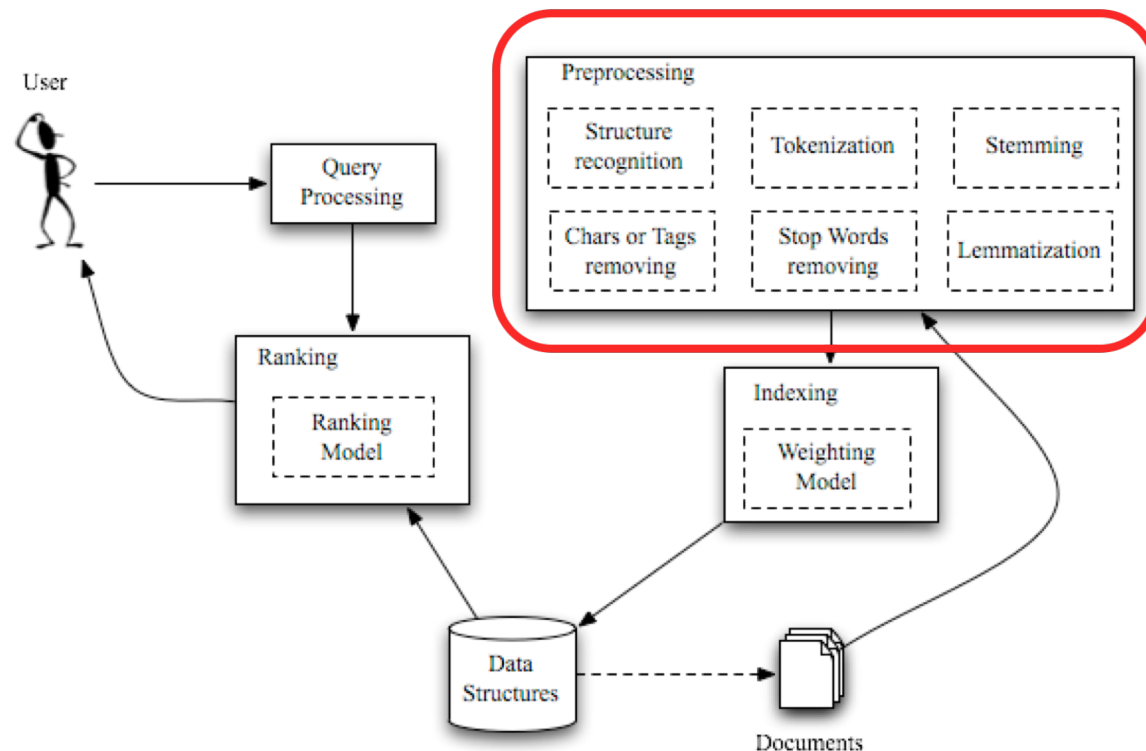
Structure of the Lucene Project



- `org.apache.lucene.search`: it contains the classes implementing different search models and ranking functions
- `org.apache.lucene.store`: it contains the classes needed to read and write to/from an index

analysis package

- It contains all functionalities required to pre-process documents
- Main functionalities
 - Tokenization.
 - Stop words removal
 - Special character removal
 - Stemming
 - Porter Stemmer
 - Snowball Stemmer



analysis package (2)



■ Pro:

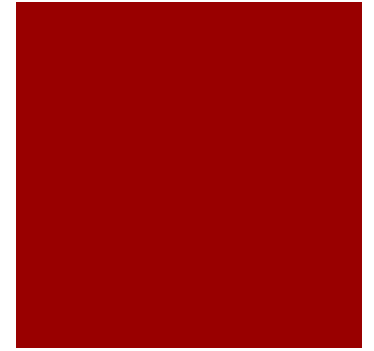
- We can customize the combination of pre-processing steps
- We can customize and (re)implement each step
 - E.g. loading different stop word lists

■ Cons:

- Intrinsic structure of a document is ignored
 - A document can be divided in “title”, “abstract”, “author”
 - The implementation of such structure is left to the programmer and it must be always re-implemented

search package

- Several query implementations have been implemented in Lucene:
 - Term Query
 - Boolean Query
 - Wildcard Query
 - Phrase Query
 - Prefix Query
 - Fuzzy Query
 - Range Query
 - Span Query



Query syntax

- Generic Query
 - pink panther
- Phrase Query
 - "pink panther"
- Boolean Query
 - "pink panther" AND return
 - "pink panther" +return
- Field specific query
 - title:"pink panther"
- Wildcard
 - pant?er
 - panther*
- Fuzzy
 - panther~
 - panther~0.8
- Boosting of a query term
 - pink panther^4
- Range
 - mod_date:[20070101 TO 20071001]

Ranking Function

(THIS IS OLD – NOW IT ADOPTS BM25)

- The final ranking function in Lucene is based on a TF * IDF weighting schema
- Given a query q , the ranking score assigned to a document d is

$$\text{sim}(q, d) = \text{coord}(q, d) \cdot \text{queryNorm}(q) \cdot \sum_{t \in q} (\text{tf}(t, q) \cdot \text{idf}(t)^2 \cdot \text{boost}(t) \cdot \text{norm}(t, d))$$

A sort of similarity $\text{sim}(\cdot, \cdot)$ between a query and a document in a Vector Space Model



Ranking Function (2)

$$\text{sim}(q, d) = \text{coord}(q, d) \cdot \text{queryNorm}(q) \cdot \sum_{t \in q} (\text{tf}(t, d) \cdot \text{idf}(t)^2 \cdot \text{boost}(t) \cdot \text{norm}(t, d))$$

$\text{coord}(q, d)$ It counts the number of terms from q occurring in d

$\text{boost}(t)$ The boost (weight) associated to each query term

$\text{tf}(t, q)$ $\sqrt{\#t_d}$

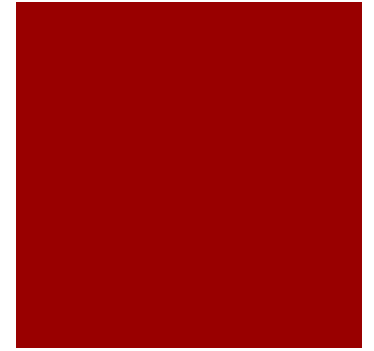
$\text{idf}(t)$ $1 + \log \left(\frac{\#D}{\text{docFreq}(t)} \right)$

$\text{queryNorm}(q)$ $\frac{1}{\text{boost}(q) \cdot \sum_{t \in q} (\text{idf}(t) \cdot \text{boost}(t))}$

$\text{norm}(t, d)$ Normalize the importance document/query in the collection, e.g. considering the document length

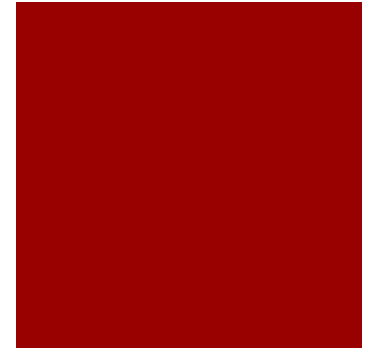
Indexing procedure

- Define a sequence of pre-processing steps, extending the `Analyzer` class
- For each document in the collection
 - Build a `Document` object populating its `Fields`
 - Define an `IndexWriter` and adding the document to the index by using the `addDocument()` method



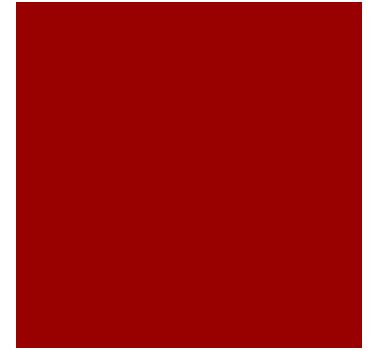
Search procedure

- Adopt the same pre-processing steps used in indexing
- Access the document index using the class `IndexSearcher`
- Define the module that analyze the query (`QueryParser`)
 - Define the `Query` model to be adopted
- Use the `IndexSearcher` to retrieve the documents



Luke

- **Luke** is a graphical user interface (GUI) to access a Lucene index and execute the following operations:
 - Browse the indexed documents and their corresponding fields
 - Execute (complex) queries
 - Modify the index
 - E.g. deleting some documents
 - ...
- It can be downloaded from
 - <http://www.getopt.org/luke/>



File Tools Settings Help

Overview Documents Search Files Plugins

Index name: /Users/diego/Desktop/DL/lucene-2.2.0/index

Number of fields: 3

Number of documents: 1849

Number of terms: 32234

Has deletions?: No

Index version: 1192659700134

Last modified: Thu Oct 18 00:24:48 CEST 2007

Directory implementation: org.apache.lucene.store.FSDirectory

Re-open

Close

Select fields from the list below, and press button to view top terms in these fields. No selection means all fields.

Available Fields

<contents>

<modified>

<path>

Show top terms >>

Number of top terms:

50

Hint: use Shift-Click to select ranges, or Ctrl-Click to select multiple fields (or unselect all).

Top ranking terms. (Right-click for more options)

No	Rank	Field	Text
1	1793	<contents>	text
2	1793	<contents>	0
3	1790	<contents>	1
4	1790	<contents>	size
5	1786	<contents>	name
6	1785	<contents>	type
7	1785	<contents>	link
8	1783	<contents>	http
9	1782	<contents>	title
10	1779	<contents>	apache
11	1773	<contents>	font
12	1769	<contents>	border
13	1769	<contents>	style
14	1769	<contents>	en
15	1769	<contents>	html
16	1768	<contents>	width
17	1767	<contents>	href
18	1767	<contents>	body
19	1766	<contents>	css
20	1766	<contents>	stylesheet
21	1763	<contents>	16
22	1762	<contents>	class
23	1761	<contents>	lucene

Index name: /Users/diego/Desktop/DL/lucene-2.2.0/index

Browse by doc. number:

Doc. #: 0 1083 1848

Browse by term:

(Hint: enter a substring and press Next to start at the nearest term).

Term: contents

Doc freq of this term: 2

Document: ? of 2

Term freq in this doc: ?

Doc #: 1083 , document boost: 1.0

Flags: I - Indexed; T - Tokenized; S - Stored; V - Term Vector (o - offsets; p - positions)
O - Omit Norms; L - Lazy; B - Binary; C - Compressed

Field	ITSVopOLBC	Boost	String Value
<contents>	-----		<not available>
<modified>	I-S-----	1.0	200706162117
<path>	I-S-----	1.0	docs/api/org/apache/lucene/index/IndexModifier.html

Copy text to Clipboard:

Enter search expression here:

indexreader

Query details: Update Explain structure

contents:indexreader

Parsed

Rewritten

 Return all matching results, even low-scored

Search

Analyzer to use for query parsing:

NOTE: use fully-qualified class name here.

org.apache.lucene.analysis.KeywordAnalyzer

Default field is: contents

SnowballAnalyzer name:

(Expert) Similarity implementation:

 Default Similarity Current custom Similarity [Design...](#)

Current: org.getopt.luke.plugins.CustomSimilarity

Results: (Hint: Double-click on results to display all fields)

Explain

Delete

148 doc(s)

0-19



#	Score	Doc. Id	contents	modified	path
0	0,4066	1058		2007061621	docs/api/org/apache/lucene/index/class-use/IndexReader.html
1	0,3194	1079		2007061621	docs/api/org/apache/lucene/index/FilterIndexReader.html
2	0,2961	1107		2007061621	docs/api/org/apache/lucene/index/package-use.html
3	0,2857	1103		2007061621	docs/api/org/apache/lucene/index/MultiReader.html
4	0,2617	1108		2007061621	docs/api/org/apache/lucene/index/ParallelReader.html
5	0,2474	1354		2007061621	docs/api/org/apache/lucene/search/function/class-use/DocValues.html
6	0,2369	1046		2007061621	docs/api/org/apache/lucene/index/class-use/CorruptIndexException.html
7	0,2357	1338		2007061621	docs/api/org/apache/lucene/search/FieldCache.html
8	0,2182	1505		2007061621	docs/api/org/apache/lucene/search/spans/class-use/Spans.html
9	0,2151	1065		2007061621	docs/api/org/apache/lucene/index/class-use/StaleReaderException.html
10	0,2090	1085		2007061621	docs/api/org/apache/lucene/index/IndexReader.html
11	0,2020	1542		2007061621	docs/api/org/apache/lucene/search/Weight.html
12	0,1924	1216		2007061621	docs/api/org/apache/lucene/queryParser/surround/query/class-use/SimpleTer
13	0,1844	1267		2007061621	docs/api/org/apache/lucene/search/class-use/FieldCache.html
14	0,1734	1430		2007061621	docs/api/org/apache/lucene/search/IndexSearcher.html
15	0,1683	1105		2007061621	docs/api/org/apache/lucene/index/package-summary.html
16	0,1683	1424		2007061621	docs/api/org/apache/lucene/search/highlight/TokenSources.html
17	0,1666	1054		2007061621	docs/api/org/apache/lucene/index/class-use/IndexDeletionPolicy.html
18	0,1649	325		2007061621	docs/api/org/apache/lucene/benchmark/byTask/PerfRunData.html

Enter search expression here:

indexreader class document

Analyzer to use for query parsing:

NOTE: use fully-qualified class name here.

org.apache.lucene.analysis.KeywordAnalyzer

Default field is: contents

SnowballAnalyzer name:

(Expert) Similarity implementation:

Default Similarity

Current custom Similarity Design...

Query details: Update Explain structure

contents:indexreader c

Query Structure

Structure of the query:

```

BooleanQuery: boost=1,0000
├─ clauses=3, maxClauses=1024, scorer14=false
│   └─ Clause 0:
│       └─ TermQuery: boost=1,0000
│           └─ Term: field='contents' text='indexreader'
│   └─ Clause 1:
│       └─ TermQuery: boost=1,0000
│           └─ Term: field='contents' text='class'
│   └─ Clause 2:
│       └─ TermQuery: boost=1,0000
│           └─ Term: field='contents' text='document'
    
```

OK

Results: (Hint: Double-cl

#	Score	Doc. Id
0	0,1536	
1	0,1536	
2	0,0167	
3	0,0186	
4	0,0146	
5	0,0164	
6	0,0164	
7	0,0193	
8	0,0470	
9	0,0753	
10	0,0899	
11	0,2591	11
12	0,1063	12
13	0,0209	13
14	0,0174	14
15	0,0146	15
16	0,0164	16
17	0,0164	17
18	0,0193	18

2007061621docs/api/index-all.html
2007061621docs/api/index.html
2007061621docs/api/lucli/class-use/Lucli.html
2007061621docs/api/lucli/Lucli.html
2007061621docs/api/lucli/package-frame.html
2007061621docs/api/lucli/package-summary.html
2007061621docs/api/lucli/package-tree.html
2007061621docs/api/lucli/package-use.html

Enter search expression here:

indexreader class document

Analyzer to use for query parsing:

NOTE: use fully-qualified class name here.

org.apache.lucene.analysis.KeywordAnalyzer

Default field is: contents SnowballAnalyzer name:

(Expert) Similarity implementation:
 Default Similarity
 Current custom Similarity [Design...](#)
 Current: org.getopt.luke.plugins.CustomSimilarity

Query details: Update Explain structure

contents:indexreader contents:class contents:document Parsed
 Rewritten

Return all matching results, even low-scored

Results: (Hint: Double-click on result)

#	Score	Doc. Id	content
0	0,1536	0	
1	0,1536	1	
2	0,0167	2	
3	0,0186	3	
4	0,0146	4	
5	0,0164	5	
6	0,0164	6	
7	0,0193	7	
8	0,0470	8	
9	0,0753	9	
10	0,0899	10	
11	0,2591	11	
12	0,1063	12	
13	0,0209	13	
14	0,0174	14	
15	0,0146	15	
16	0,0164	16	
17	0,0164	17	
18	0,0193	18	

Explanation

Explanation of the document hit:

- 0,1536 sum of:
 - 0,0258 weight(contents:indexreader in 1), product of:
 - 0,7511 queryWeight(contents:indexreader), product of:
 - 3,5185 idf(docFreq=148)
 - 0,2135 queryNorm
 - 0,0344 fieldWeight(contents:indexreader in 1), product of:
 - 1,0000 tf(termFreq(contents:indexreader)=1)
 - 3,5185 idf(docFreq=148)

OK

1762 doc(s) 0-19

Extractor.html

nl

nl



Solr

- Solr is a standalone enterprise search server implemented from Apache
 - Solr is built using the API provided by Lucene
- Solr implements REST-like API
- It supports different standard to represent query and retrieve documents
 - JSON and XML are the more used
- It implements a Web-based Administration console
- It allow to distribute and replicate indexes on different machines
- <http://lucene.apache.org/solr/>



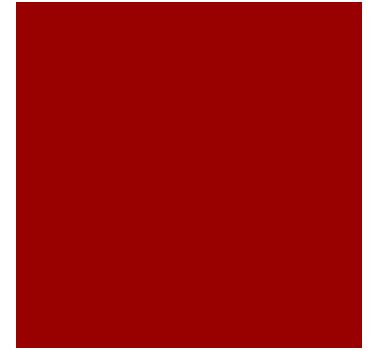
Lemur



- Lemur is another framework for the development of IR systems
- It is the result of a collaboration between
 - *Computer Science Department at the University of Massachusetts*
 - *School of Computer Science at Carnegie Mellon University.*
- The core of Lemur is implemented in C++
- <http://www.lemurproject.org/>
- <http://sourceforge.net/p/lemur/wiki/Home/>

Exercise Objectives

- Development of a simple IR system, based on Lucene that supports
 - the indexing of documents
 - the retrieval of documents
- Evaluation of the resulting system



You will be provided with...



- A small dataset, the Cranfield collection, made of 1398 abstracts
 - http://ir.dcs.gla.ac.uk/resources/test_collections/cran/
- In the folder `resources/cranfield_collection splitted` you can find the documents separated in different files
- Each document has the following format

```
.T
Document_title
.A
Authors
.B
Bibliographic_References
.W
Text
```

An example of document



.T

experimental investigation of the aerodynamics of a wing in a slipstream .

.A

brenckman,m.

.B

j. ae. scs. 25, 1958, 324.

.W

experimental investigation of the aerodynamics of a wing in a slipstream .

an experimental study of a wing in a propeller slipstream was made in order to determine the spanwise distribution of the lift increase due to slipstream at different angles of attack of the wing and at different free stream to slipstream velocity ratios .

...

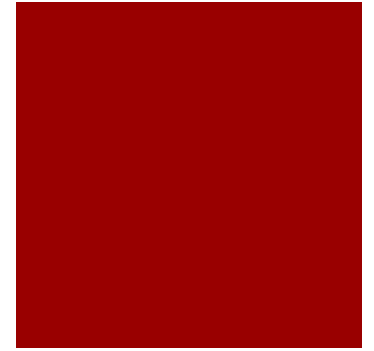
How to evaluate the IR system



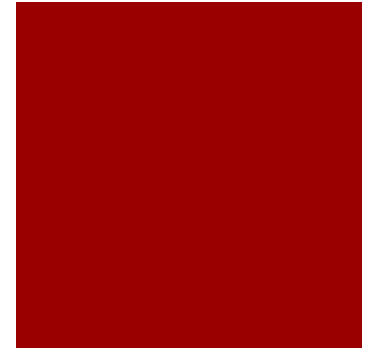
- The file `resources/cranfield_query/cran_query.text` contains 225 query for the system
- The file `resources/cranfield_query/qrels.text` contains, for each query, the set of relevant document that the system is expected to retrieve
- For example, the query 1:
 - *"what similarity laws must be obeyed when constructing aeroelastic models of heated high speed aircraft ."*
- should retrieve the following document
 - 184, 29, 31, 12, 51, 102, 13, 14, 15, 57, 378, 859, 185, 30, 37, 52, 142, 195, 875, 56, 66, 95, 462, 497, 858, 876, 879, 880, 486

Exercise

- Create two indexes from the Cranfield corpus by considering two different pre-processors
- You are provided of the JAVA code implementing the indexing and retrieval phase
- For each query in the dataset, retrieve the document from each system and measure
 - Precision
 - Recall
 - F1



Exercise facilities



- In the folder `code`
- To compile the search engine based on lucene
 - `bash compile.sh`
- To index the Cranfield corpus
 - `bash build_index.sh`
- To execute Luke
 - `bash launch_luke.sh`
- To retrieve documents
 - `python evaluate_search.py`

Exercise



- You can find a set of queries in the file
`code/resources/cranfield_query/cran_query.text`
- For each query the set of document to be retrieved is reported in
`code/resources/cranfield_query/qrels.text`
- If you are not expert with JAVA, you are also provided with a python script to evaluate Input queries
- Measure
 - precision, recall and F1 by using the gold standard
 - MAP
- Modify the indexing process to improve F1