Information Retrieval Exercises

Assignment 3:

Boolean Information Retrieval with Lucene

Mario Sänger (saengema@informatik.hu-berlin.de)
Lucene

- Java-based information retrieval engine
  - Apache Open Source Project
  - Widespread library for full text search
  - Related projects: ElasticSearch, Solr, Tika, Nutch, ...

- We will use the core library of Lucene!

- Requires two steps:
  - Indexing: Create a Lucene index on the documents
  - Searching: Parse a query and lookup the index
Task

• Implement Boolean IR as in assignment 2, but this time use Lucene:
  – Parse the IMDB movie plots
  – Treat all text in lowercase for indexing and searching
  – Use word tokenization and stop word removal, but no stemming

• You can reuse your plot file parser from Assignment 2!
• But use Lucene (v7.3.1) for indexing and searching!
  – http://www.apache.org/dyn/closer.lua/lucene/java/7.3.1
Lucene: Basic concepts

- Lucene builds inverted indices and allows queries on these indices
- A Document is the unit of search and index
  - A document consists of one or more fields
  - A field is a key-value pair
- Indexing involves adding documents to an IndexWriter
- Searching involves retrieving documents via an IndexSearcher
Lucene: Basic concepts

- **Tokenizers**: break field data into lexical units, or tokens
- **Filters**: examine a stream of tokens and keep them, transform or discard them, or create new ones
- **Analyzers**: tokenizers and filters may be combined
  - Combination is called an analyzer
  - The output of an analyzer is used to query or build indices

- Use the same analyzer for querying and building indices!
Lucene: Analyzers

- Lucene provides multiple default tokenizers, i.e.:
  - LetterTokenizer: divide text at non-characters
  - WhiteSpaceTokenizer: divide text at whitespace characters
  - StandardTokenizer: grammar-based tokenizer

- Lucene provides multiple default filters, i.e.:
  - LowerCaseFilter: converts any uppercase letters to lowercase
  - Word Stemming filters (Kstem, Hunspell, Snowball Porter, ...)

- Lucene provides multiple default analyzers, i.e.:
  - SimpleAnalyzer: LetterTokenizer, LowerCaseFilter
  - StandardAnalyzer: StandardTokenizer, LowerCaseFilter, English stop words
  - WhiteSpaceAnalyzer: WhiteSpaceTokenizer
  - StopAnalyzer: LetterTokenizer, LowerCaseFilter, English stop words
Lucene API: Indexing

- Specify the analyzer to use
  
  Analyzer myAnalyzer = new StandardAnalyzer(); // or another Analyzer!

- Specify a directory and an index writer
  
  Directory index = FSDirectory.open(Paths.get(directory));
  IndexWriterConfig config = new IndexWriterConfig(myAnalyzer);
  IndexWriter writer = new IndexWriter(index, config);

- Create a document and add this document to the index:
  
  Document doc = new Document();
  doc.add(new StringField("id", id, Field.Store.YES));
  doc.add(new TextField("title", title, Field.Store.YES));
  writer.addDocument(doc);

- Close index writer:
  
  writer.commit()
  writer.close();
Lucene API: Field types

- Fields types for text:
  - TextFields will be tokenized. Used for texts that needs to be tokenized
  - StringFields will be treated as a single term. Used for atomic values that are not to be tokenized

- Many other typed fields:
  - IntPoint/LongPoint: int/long indexed for exact/range queries
  - FloatPoint/DoublePoint: float/double indexed for exact/range queries

- Field.Store.YES: indexed & returned as result
- Field.Store.NO: indexed but not returned as result
Lucene API: Querying

- Open Lucene index for searching
  
  ```java
  IndexReader indexReader = DirectoryReader.open(index);
  IndexSearcher indexSearcher = new IndexSearcher(indexReader);
  ```

- Parse title:`<querystr>` using the analyzer
  
  ```java
  Query query = new QueryParser("title", myAnalyzer).parse(querystr);
  ```

- Retrieve all results
  
  ```java
  TopDocs hits = indexSearcher.search(query, Integer.MAX_VALUE);
  
  long totalHits = hits.totalHits;
  for (ScoreDoc result: hits.scoreDocs) {
      Document document = indexReader.document(result.doc);
  }
  ```
Lucene Query Parser syntax

• You have to support the Query Parser syntax:
  – term query syntax:
    title:Game
  – phrase query syntax:
    title:”Game of Thrones”
  – AND query, OR query
    title:”Game of Thrones” AND (plot:Baelish OR plot:Jon)
  – More features: NOT queries, wildcards, proximity, range searches, fuzzy searches, regular expressions, ...

• There is a built-in Query Parser for this in Lucene!
Example queries

1. title:"game of thrones" AND type:episode AND (plot:Bastards OR (plot:Jon AND plot:Snow)) -plot:son
3. plot:Berlin AND plot:wall AND type:television
4. plot:men~1 AND plot:women~1 AND plot:love AND plot:fool AND type:movie
5. title:westworld AND type:episode AND year:2016 AND plot:Dolores
Example queries

7. plot:Hero AND plot:Villain AND plot:destroy AND type:movie


10. plot:Hero AND plot:Marvel -plot:DC AND type:movie

11. plot:Hero AND plot:DC -plot:Marvel AND type:movie
Searchable fields

• Searchable fields are as follows:
  – title
  – plot (if a document has multiple plot descriptions they can be appended)
  – type (movie, series, episode, television, video, videogame; see next slide)
  – year (optional)
  – episodetitle (optional, only for episodes)

• There is a built-in MultiFieldQueryParser for this in Lucene!
Revised: Movie corpora

- Reuse the corpus plot.list
  - Plain text, roughly 400 MB

- Supported document types and their syntax in the corpus:
  - movie: MV: <title> (<year>)
  - series: MV: "<title>" (<year>)
  - episode: MV: "<title>" (<year>) {<episodetitle>}
  - television: MV: <title> (<year>) (TV)
  - video: MV: <title> (<year>) (V)
  - videogame: MV: <title> (<year>) (VG)
Getting started

• Get Apache Lucene v7.3.1 core and queryparser library
  – ... via Download

  http://archive.apache.org/dist/lucene/java/7.3.1/

  – ... via Maven

  `<dependency>
      <groupId>org.apache.lucene</groupId>
      <artifactId>lucene-core</artifactId>
      <version>7.3.1</version>
  </dependency>

  `<dependency>
      <groupId>org.apache.lucene</groupId>
      <artifactId>lucene-queryparser</artifactId>
      <version>7.3.1</version>
  </dependency>`
Preprocessing

- Indexing: the corpus text has to be tokenized.
- Search: the query has to be tokenized, too.
- Convert all words to lower case (case-insensitive search and indexing) and remove English stop words.
- There are built-in “Analyzers” for this in Lucene.
Implementation

- We provide a class skeleton: BooleanSearchLucene.java

- public void buildIndices(Path plotFile)
  - Used to parse the file and build the Lucene index

- public Set<String> booleanQuery(String queryString)
  - Parses the query string and returns the title lines of any entries in the plot file matching the query

- public void close()
  - Free used resources (e.g. close Lucene index, thread pools)
Test your program

- We provide you with:
  - `queries_lucene.txt`: file containing exemplary queries
  - `results_lucene.txt`: file containing the expected results of running these queries
  - a main method for testing your code (which expects as parameters the corpus file, the queries file and the results file)
Submission

- **Group 1:** Friday, 15.06., 23:59 (midnight)
- **Group 2:** Sunday, 17.06., 23:59 (midnight)

- Submit a ZIP archive named `ass3_<group-name>.zip`
  - Java source files of your solution
  - Compiled and executable `BooleanQueryLucene.jar`

- Upload archive to the HU-BOX:
  [https://box.hu-berlin.de/u/d/e5f31199fe864ff6a4ec/](https://box.hu-berlin.de/u/d/e5f31199fe864ff6a4ec/)
Submission requirements

- Test your jar before submitting by running the examples queries on one of the gruenau hosts
  - `java -jar BooleanQueryLucene.jar <plot list file> <queries file> <results file>`
  - You might have to increase the JVM’s heap size (e.g., `-Xmx8g`)
  - Your jar must run and answer all test queries correctly!

- Your program has to correctly answer all example queries correctly to pass the assignment!
Solution presentations

• The presentation of the solutions will be given on 25.06. resp. 27.06

• You are be able to pick when and what you’d like to present (first-come-first-served):
  – Group 1 (Mo): https://dudle.inf.tu-dresden.de/ir_ass3_mo/
  – Group 2 (We): https://dudle.inf.tu-dresden.de/ir_ass3_we/

• Presentation of the following aspects:
  – Indexing implementation
  – Query implementation
Competition

- Index as fast as possible
- Note that everybody uses the same indexer (Lucene)

- Look for possible optimizations
  - For example: http://www.lucenetutorial.com/lucene-nrt-hello-world.html

- Stay under 50 GB memory usage
- We will call the program using our evaluation tool:
  - We will use different queries and -Xmx50g parameter
Submission checklist

1. Did not change or remove any code from BooleanQueryLucene.java
2. Did not alter the functions` signatures (types of parameters, return values)
3. Only use the default constructor and don`t change its parameters
4. Did not change the class or package name
5. Named your jar BooleanQueryLucene.jar
6. Tested your jar on gruenau hosts by running
   java -jar BooleanQueryLucene.jar plot.list queries.txt results.txt
   (you might have to increase Java heap space, e.g. -Xmx6g)
7. Ascertained that the queries in queries_lucene.txt were answered correctly
8. Make sure to upload a zip file named by your group name.
Timetable / Next steps

- **Assignment 3 submission deadline:**
  - **Group 1:** Friday, 15.06., 23:59 (midnight)
  - **Group 2:** Sunday, 17.06., 23:59 (midnight)

- **Presentations of the solutions for assignment 2**
  - **Group 1:** Monday, 11.06.
  - **Group 2:** Wednesday, 13.06