Information Retrieval

Assignment 2: Boolean Information Retrieval

Patrick Schäfer (patrick.schaefer@hu-berlin.de)

Marc Bux (buxmarcn@informatik.hu-berlin.de)

Assignment 2: Boolean IR on Full Text

- last assignment: hard-coded queries on small data
- this time: arbitrary queries on large(r) data
- obtain corpus plot.list from http://www.imdb.com/interfaces
  - read license: only for personal use, do not redistribute
  - plain text, roughly 400 MB
  - updated version every Friday
  - you can reduce the size of the corpus by using the HEAD and TAIL tools:
    head -n 10000 plot.list > small.list
- write a program which can find
  - arbitrary terms,
  - arbitrary phrases, and
  - arbitrary combinations thereof
- in other words: implement Boolean IR
  - term, phrase and AND search

Example (Corpus Excerpt)

MV: Moonraker (1979)

PL: James Bond is back for another mission and this time, he is blasting off
PL: into space. A spaceship traveling through space is mysteriously hi-jacked
PL: and Bond must work quickly to find out who was behind it all. He starts
PL: with the rockets creators, Drax Industries and the man behind the
PL: organisation, Hugo Drax. On his journey he ends up meeting Dr. Holly
PL: Goodhead and encounters the metal-toothed Jaws once again.

BY: simon

PL: A Boeing 747 carrying a US space shuttle on loan to the UK crashes into the
PL: Atlantic Ocean. When the British examine the wreckage they can find no
PL: trace of the spacecraft and send agent James Bond to the shuttle's
PL: manufacturers, Drax Industries, to investigate.

BY: Dave Jenkins

Query Syntax

- token query syntax: <field>:<token>
- phrase query syntax: <field>:"<phrase>"
- and query syntax: <query> AND <query>
  (where <query> can be a token, phrase, or AND query)
- 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)
- the syntax is a subset of the Lucene QueryParser syntax
  - http://www.lucenetutorial.com/lucene-query-syntax.html
- “ AND ” and double quotes not allowed in tokens or phrases
  - i.e., don’t worry about queries like *title:"BATMAN AND ROBIN"*

Corpus

- the file contains documents from IMDB
- 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)
- corpus is in ISO-8859-1 format

  -
    ```
    BufferedReader reader =  new BufferedReader(
      new InputStreamReader(new FileInputStream(path),
        StandardCharsets.ISO_8859_1));
    ```

Documents

- an entry starts with “MV: ” and ends with horizontal lines (“---”) or at the end of the corpus
- each entry must be treated as one document
  - which may match a query or not
  - identified by their full title line in the corpus: e.g., *MV: Moonraker (1979)*
- again, every document has up to five searchable fields: title, plot, type, year, episodetitle
- other information (e.g., “BY: “) in the corpus can be discarded

Some Peculiar Documents

- *MV: Disparity (2013) {{SUSPENDED}}*
  MV: "Moments" (2011) {Dreams (#1.1)} {{SUSPENDED}}
  → {{SUSPENDED}} can be discarded
- *MV: Disparity (????)*
  → not all entries have a year field
- *MV: Displaced (2014/II)*
  MV: Displaced (2014/III)
  MV: The Ambassador (????/IV)
  MV: The Ambassador (1984)
  → different documents may have identical name, year, and type
- *MV: Þegar það gerist (1998) (TV)*
  → make sure to parse the file using ISO-8859-1 encoding

Preprocessing

- corpus text has to split (“tokenized”) into terms to build indices
- in phrase search, the query is a consecutive sequence of terms
- convert terms (for indices, term queries, and phrase queries) to lower case (case-insensitive search)
- use as delimiters only blanks, dots, commas, colons, exclamation marks, and question marks (.,:!?)
- leave all other special characters untouched
  - they become parts of tokens
- examples:
  - “The Lord of the Rings: The Two Towers”
    → “the”, “lord”, “of”, “the”, “rings”, “the”, “two”, “towers”
  - “Marvel’s The Avengers” → “marvel’s”, “the”, “avengers”

Program

- in BooleanSeach.java, implement the functions for building indices and running queries:
  - public void buildIndices(String plotFile)
  - public Set<String> booleanQuery(String queryString)
    (returns the title lines of any entries in the plotFile matching the query)
  - for more detailed information, consult Javadoc for both methods
  - only add classes and code, do not change or remove any code
  - do not alter the functions‘ signatures (types of params, return values)
  - do not change the class or package name
  - only use the default constructor and don‘t change its parameters
- do not use Lucene (yet)
  - you may use other libraries

Test your Program

- we provide you with:
  - a queries.txt file containing exemplary queries
  - a results.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)

- additionally, you can write your own test queries
  - check the plausibility of your results using GREP:
    grep " <search-token> " <corpus-file>
  - use -G or -P parameter for regular expressions

Example Queries (from queries.txt)

- *title:"game of thrones" AND type:episode AND plot:shae AND plot:Baelish*
- *plot:Skywalker AND type:series*
- *plot:"year 2200"*
- *plot:Berlin AND plot:wall AND type:television*
- *plot:Cthulhu*
- *title:"saber rider" AND plot:april*
- *plot:"James Bond" AND plot:"Jaws" AND type:movie*
- *title:"Pimp my Ride" AND episodetitle:mustang*
- *plot:"matt berninger"*
- *title:"grand theft auto" AND type:videogame*
- *plot:"Jim Jefferies"*
- *plot:Berlin AND type:videogame*
- *plot:starcraft AND type:movie*
- *type:video AND title:"from dusk till dawn"*

Deliverables

- by Thursday, 8.12., 23:59 (midnight)
  - two-and-a-half weeks
- submission: archive (zip, tar.gz)
  - contains Java source files, any used libraries, and your compiled jar named BooleanQuery.jar
  - file name (of submitted archive): your group name
- upload to https://box.hu-berlin.de/u/d/76b81341a0/
  - if this doesn't work, send via mail to buxmarcn@informatik.hu-berlin.de
- test your jar before submitting by running our queries on gruenau2
  - java -jar BooleanQuery.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

Presentation of Solutions

- you are be able to pick when and what you‘d like to present (first-come-first-served):
  - monday: https://dudle.inf.tu-dresden.de/inforet_ue2_mo/
  - tuesday: https://dudle.inf.tu-dresden.de/inforet_ue2_tu/
- presentation will be given on 12./13.12.

- three teams will present their term search & indices
- one team will present their phrase search
- one team will present their AND search
- one team will present their parser

Competition

- search as fast as possible
- build as many indices as you deem necessary
  - stay under 50 GB memory usage
- we will call the program using our eval tool
  - we will use 9 different queries and -Xmx50g parameter
- the time for building the index counts as much as a single query
  - i.e., one tenth of the total achievable competition points

Challenges

- parse "indexable" documents from an unstructured text file
  - handle special characters, unexpected syntax variants
- conceptualize and implement indices
  - size will not be evaluated
  - for different fields (title, plot, year, type, episodetitle)?
- efficient computation of document lists per term
  - might be large (e.g., searching for "the")
- efficient implementation of AND operator
  - fast intersection of document lists
- efficient implementation of evaluating entire query
- implementation of phrase search

Checklist

again, before submitting your results, make sure that you

1. did not change or remove any code from BooleanQuery.java
2. did not alter the functions‘ signatures (types of params, 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 BooleanQuery.jar
6. tested your jar on gruenau2 by running
   java -jar BooleanQuery.jar plot.list queries.txt results.txt
   (you might have to increase Java heap space, e.g. -Xmx6g)
7. ascertained that the 15 queries in queries.txt were answered correctly

FAQ

1. May we use a database for the indices?
   – Yes, but only an embedded one (SQLite etc.).
2. May we create more than one index?
   – Yes, as long as you stay below 50 GB memory usage.
3. What is the specification of the evaluation VM?
   – 50 GB memory, 15 virtual cores, 200 GB HDD.
4. Should we parse and index the number of a series’ episode? (e.g., season 3, episode 6 in {The Climb (#3.6)})
   – No, treat all content in the curly braces (including the episode number) as text, which can then be queried via the field “episodetitle”.
5. Are phrase queries case-insensitive?
   – Yes, all queries (and thereofre also the indices) are case-insensitive.

FAQ (2)

6. Queries are case-insensitive and therefore the indices should be case-insensitve. Does that imply that the title lines returned by the booleanQuery method may be in lower case as well?
   - No, the strings returned by the booleanQuery method ought to be identical to the title lines in the corpus.
7. Are semicolons also a valid delimiter when determining terms from text (i.e., during tokenization)?
   - No, the only valid delimiters are blanks, dots, commas, colons, exclamation marks, and question marks. This implies that the text “obsession: party;” results in the terms “obsession” “party;”.
8. Does our BooleanQuery program have access to a local copy of the corpus or will it have to obtain the corpus from the web?
   - The buildIndices method expects the corpus to be accessible locally under the path indicated by plotFile parameter.

FAQ (3)

9. How can the parser decide if a document is a movie or series?
   – Series have their title enclosed by double quotes, movies don‘t. See slide 5 for an overview of the supported types and their syntax. See https://contribute.imdb.com/updates/guide/title_formats for even more information on formats.
10. Some documents have a version number besides their year, such as MV: Displaced (2014/II). Should our tool be able to query this version number?
    – The field "year" should merely allow for querying the year itself. The version number (in this case "II") is not part of the year and we won’t test your program with queries like *year:II*. Note however that documents whose title lines are exactly identical besides the version number, should also be treated (and returned) as two separate documents.

FAQ (4)

11. Can you concretize phrase query?

   – Here, we define a phrase as a consecutive sequence of terms. You can determine the terms of a corpus via tokenization (i.e., by splitting at any of the delimiters [.,:!?]) and converting to lower case. You can treat phrase queries in exactly the same way. Effectively, a document matches a phrase query if it contains all of the query's terms in exactly the same order as in the query.
   For instance, a document with plot *the cat, the "dog"* should match queries *plot:"cat, the"*, *plot:"cat the"*, *plot:"the,,,:::cAt THE"* ([.,:!?] are all delimiters). However, it should not match the query *plot:"he cat"* (because the term "he" is not contained in the document and phrase queries, as we define them, are not the same as substring searches). Whether or not it matches the query *plot:"the "dog""* is undefined (double quotes are not interpreted as a delimiter and we don’t allow AND or double quotes in queries anyways).

FAQ (5)

12. Should we implement the search such that "????" is a valid query for the year field?
    - No, the field "year" is optional and documents with an undefined year (i.e., "????") don't have to be queryable via their year.
13. A lot of the documents have a series and episode number at the end of their episodetitle field, e.g. *{Dark Wings, Dark Words (#3.2)}*. Should we parse and index this series and episode number?
    - No, just ignore it and interpret all content in the curly braces as the episodetitle. If your tokenization works as intended, it will split the string *(#3.2)* into terms *(#3* and *2)*. If someone wanted to query series or episode number, they could then do so by formulating the query *episodetitle:(#3* .

FAQ (6)

14. Does the corpus contain documents with brackets in their title?
    - Yes. Running the command
      grep -P "MV: \".\(.*\" \(" plot.list*
      reveals document like
      MV: "4X (Non-verbal)" (2012) .
15. During evaluation, is there a time limit for building the index ?
    - Yes. Since we have to evaluate more than 20 submissions over the course of a few days, there is only so much time we can allot to each submission. Specifically, if your index does not build within 60 minutes, we will terminate the process. In this case, all queries will count as “failed”.
      Also, if any of your queries takes longer than five minutes, we will terminate that query. In this case, only that query will count as “failed”.

FAQ (7)

16. Can you tell us more about the evaluation procedure?
    - Yes. We will run nine new, potentially more difficult, queries using our own runnable Jar, which will import and invoke the methods you implemented. For each of these queries as well as for building the index, we will measure runtime and determine your rank. If your submission does not return the "correct" results, it will get the (shared) maximum rank for this particular query. To determine the total rank of your submission, we will compute the average across all ten ranks.
17. When does this assignment count as "completed"?
    - When **all** of the original 14 queries in the queries.txt file we provided are processed correctly, i.e., when they return the results listed in the results.txt file. Therefore, please check if your implementation adheres to this requirement before submitting. **If your implementation does not return the expected result for any of the 14 provided queries, it will be excluded from the contest** and you will have to provide a corrected version.

FAQ (8)

18. In my implementation, query *plot:Skywalker type:series* returns a lot more results than the ones found in results.txt, e.g., *MV: "Star Wars: Clone Wars" (2003) {Chapter 1 (#1.1)}*. Are the results in results.txt incomplete?
    - According to our definition of series and episodes, the example you mentioned is an episode (and, thus, not a series), since it has an episode title in curly braces. Please strictly follow the syntax of the six document types found on slide 5:
      movie: MV: <title> (<year>)
      series: MV: "<title>" (<year>)
      episode: MV: "<title>" (<year>) {<episodetitle>}
      Note that a document can't have more than a single type. Therefore, when we query a series, we don't expect episodes (of this series) in the returned results.

FAQ (9)

19. When reading the results.txt file, some special characters are not correctly parsed, e.g.,
    MV: Die EislÃ¤uferin (2015) (TV) instead of
    MV: Die Eisläuferin (2015) (TV)
    Why might this be?
    - Apparently, your results.txt is not encoded as Latin-1 / ISO 8859-1 or something else has gone wrong with the file. Maybe you created a new file of different encoding and copied the content of the results.txt from our website into that file. In any case, please make sure that the file is in the correct format or re-download the file from our website.

How To: Inverted Files

- simple and effective index structure for searching terms in a collection of documents
- "bag of words" approach
- instead of "docs contain terms", we use "terms appear in docs" ("inverted" index)

	term1	term2	term3
Doc1	1	0	1
Doc2	1	0	0
Doc3	0	1	1
Doc4	1	0	0
Doc5	1	1	1
Doc6	1	1	0
Doc7	0	1	0
Doc8	0	1	0

	Doc1	Doc2	Doc3	Doc4	Doc5	Doc6	Doc7	Doc8
term1	1	1	0	1	1	1	0	0
term2	0	0	1	0	1	1	1	1
term3	1	0	1	0	1	0	0	0

Building an Inverted File [Andreas Nürnberger, IR-2007]

Doc1:
Now is the time for all good men to come to the aid of their country

term	Doc
now	1
is	1
the	1
time	1
for	1
all	1
good	1
men	1
to	1
come	1
to	1
the	1
aid	1
of	1
their	1
country	1

Doc2:
It was a dark and stormy night in the country manor. The time was past midnight

term	Doc
it	2
was	2
a	2
dark	2
and	2
stormy	2
night	2
in	2
the	2
country	2
manor	2
the	2
time	2
was	2
past	2
midnight	2

Merge

term	Doc
a	2
aid	1
all	1
and	2
come	1
country	1,2
dark	2
for	1
good	1
in	2
is	1
it	2
manor	2
men	1
midnight	2
night	2
now	1
of	1
past	2
stormy	2
the	1,2
their	1
time	1,2
to	1,2
was	1,2

Boolean Retrieval

- we can now efficiently implement Boolean queries
- for each query term term$_i$, look up document list Doc$_i$ containing term$_i$
- evaluate query in the usual order:
  - term$_i$ $\wedge$ term$_j$: Doc$_i$ $\cap$ Doc$_j$
- example:
  - plot:time AND plot:past AND plot:the

    = Doc$_{\text{plot:time}}$ $\cap$ Doc$_{\text{plot:past}}$ $\cap$ Doc$_{\text{plot:the}}$

    = {1,2} $\cap$ {2} $\cap$ {1,2}

    = {2}

term	Doc
a	2
aid	1
all	1
and	2
come	1
country	1,2
dark	2
for	1
good	1
in	2
is	1
it	2
manor	2
men	1
midnight	2
night	2
now	1
of	1
past	2
stormy	2
the	1,2
their	1
time	1,2
to	1,2
was	1,2

Next Steps

- this week: we’ll send you the evaluation of assignment 1
- next week: Q/A session for assignment 2 (as every week)
  - attendance is optional