Getting started with PyLucene

CS4201: Information Retrieval and Web Search

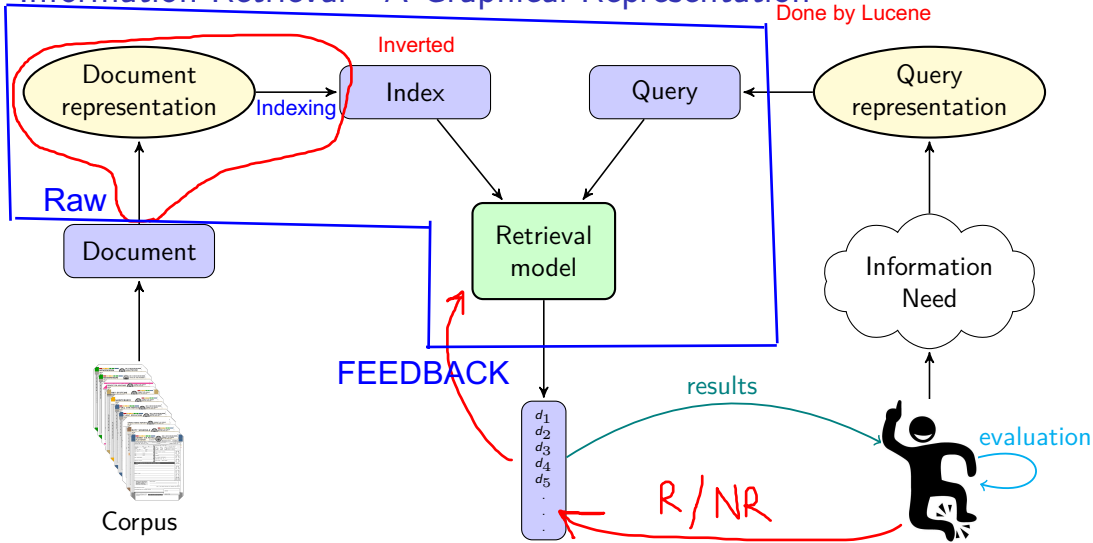
Slide courtesy

Dwaipayan Roy

IISER, Kolkata

Presented by
Sourav Saha

Information Retrieval - A Graphical Representation



Lucene efficient indexing and retrieval

Lucene (2001)

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

Background is lucene

Example

Netflix, Amazon

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!

Features of Lucene

- No built-in support for synonyums
- ~~Updating document content not allowed~~
- No built-in support for parsing regular file (such as `.doc`, `.pdf` etc.)

Features of Lucene

- No built-in support for synonyums
- ~~Updating document content not allowed~~
- No built-in support for parsing regular file (such as `.doc`, `.pdf` etc.)

Can't be used out of the box!

* We have to make it compatible

Features of Lucene

- No built-in support for synonyums
- Updating document content not allowed
- No built-in support for parsing regular file (such as `.doc`, `.pdf` etc.)

Can't be used out of the box!

Blessing in disguise!

Features of Lucene

Only Term Overlap

Car != Vehicle

- No built-in support for synonyums
- Updating document content not allowed
- No built-in support for parsing regular file (such as `.doc`, `.pdf` etc.)
- Simple API
- Fast

So amazon and other platforms use it frequently

- Concurrent indexing and searching
- A series of implemented retrieval models
- Free and open-source

* Only supports stream Writing parser is responsibility of programmer

* Multiple retrieval models are already being implemented inside lucene like BM25

Lucene for IR research

* Research Starter

- Anserini: `https://github.com/castorini/anserini`
- Lucene4IR: `https://github.com/lucene4ir`
- Luc4TREC: `https://github.com/dwaipayanroy/Luc4TREC`
- LIARR-2017: `https://liarr2017.github.io/`
- AFIRM-2019: `https://github.com/ielab/afirm2019`

Popularity of Lucene for IR Research

Retrieval Models -

- SMART (Cornell University) written in C -> 1960s
- Indri (University of Massachusetts - Amherst (UMass) and Carnegie Mellon University (CMU)) C++ -> 1990s
- Galago (UMass and CMU) JAVA -> 2010s
- Terrier (University of Glasgow) 2000s

For Academic purpose only

Popularity of Lucene for IR Research

- SMART (Cornell University)
- Indri (University of Massachusetts - Amherst (UMass) and Carnegie Mellon University (CMU))
- Galago (UMass and CMU)
- Terrier (University of Glasgow)
- Lucene (written for non-academic purposes)

Popularity of Lucene for IR Research

- SMART (Cornell University)
- Indri (University of Massachusetts - Amherst (UMass) and Carnegie Mellon University (CMU))
- Galago (UMass and CMU)
- Terrier (University of Glasgow)
- Lucene (written for non-academic purposes) - Doug Cutting -> also developed Hadoop
  - Robustness
  - Flexibility
  - Bigdata handling efficiency

Advantages

HDFS

PyLucene - a Python wrapper for Lucene

PyLucene - a Python wrapper for Lucene

- Python extension for accessing Java Lucene.

PyLucene - a Python wrapper for Lucene

- Python extension for accessing Java Lucene.
- Goal $\rightarrow$ allow us to use Lucene's *text indexing* and *searching* capabilities from Python.

PyLucene - a Python wrapper for Lucene

- Python extension for accessing Java Lucene.
- Goal $\rightarrow$ allow us to use Lucene's *text indexing* and *searching* capabilities from Python.
- Embeds a Java VM with Lucene into a Python process.

PyLucene - a Python wrapper for Lucene

- Python extension for accessing Java Lucene.
- Goal $\rightarrow$ allow us to use Lucene's *text indexing* and *searching* capabilities from Python.
- Embeds a Java VM with Lucene into a Python process.
- Built with JCC, a C++ code generator that makes it possible to call into Java classes from Python via Java's Native Invocation Interface (JNI).

PyLucene - a Python wrapper for Lucene

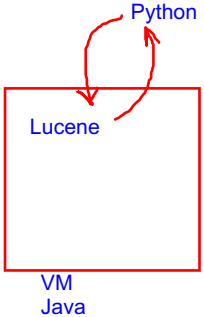
- Python extension for accessing Java Lucene.
- Goal $\rightarrow$ allow us to use Lucene's *text indexing* and *searching* capabilities from Python.
- Embeds a Java VM with Lucene into a Python process.
- Built with JCC, a C++ code generator that makes it possible to call into Java classes from Python via Java's Native Invocation Interface (JNI).
- Sources for JCC are included with the PyLucene sources.

PyLucene - a Python wrapper for Lucene

- Python extension for accessing Java Lucene.
- Goal → allow us to use Lucene's *text indexing* and *searching* capabilities from Python.
- Embeds a Java VM with Lucene into a Python process.
- Built with JCC, a C++ code generator that makes it possible to call into Java classes from Python via Java's Native Invocation Interface (JNI).
- Sources for JCC are included with the PyLucene sources.

We will be using PyLucene version ~~8.8.1~~ 9.7.0

2004 -> 256 MB RAM -> high end desktops



Installing PyLucene

Prerequisites

1. ~~Java version 1.8~~
2. ~~Ant and Ivy~~
3. Download PyLucene from: `https://www.apache.org/dyn/closer.lua/lucene/pylucene/`
4. Install JCC

Installing PyLucene

Prerequisites

1. Java version 1.8
2. Ant and Ivy
3. Download PyLucene from: `https://www.apache.org/dyn/closer.lua/lucene/pylucene/`
4. Install JCC

Contains information about installation

- Install PyLucene
  - edit `MakeFile` based on your system's Java installation paths.
  - `make` # this may need sudo permission
  - `make install` # this could take more than half an hour
  - `make test` # testing whether the installation in successful or not

Linux -> sudo -> super-user permission

Lucene document building

- Collection: set of *documents*.
- Document: set of *fields*.
- e.g. academic papers; several fields:
  - title: Relevance based Language Model
  - authors: Victor Lavrenko, Bruce Croft
  - publication-venue: SIGIR
  - year: 2001
  - pages: 120 - 127
  - keywords: information system, information retrieval, evaluation
  - abstract: We explore the relation between classical probabilistic models of information retrieval ...

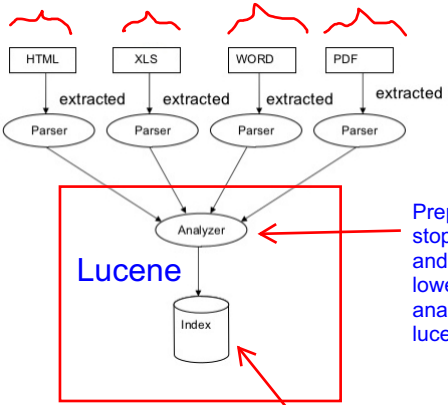
Title
Author
Abstract
Introduction
Other fields

Lucene document building

- Collection: set of *documents*. - movie information
- Document: set of *fields*. - containing different informations about movie
- e.g. movie information; several fields:
  - title: `The Dark Knight`
  - release year: 2008
  - origin: `American`
  - director: `Christopher Nolan`
  - cast: `Christian Bale, Heath Ledger, Michael Caine, Gary Oldman, Morgan Freeman...`
  - genre: `superhero`
  - plot: `A gang of criminals rob a Gotham City mob bank, murdering each other until only the mastermind remains: the Joker, who escapes with the money...`
  - wikipage: `https://en.wikipedia.org/wiki/The_Dark_Knight_(film)`

Indexing in Lucene

Collection(s) or Corpus(Corpora)



Preprocessing stopword removal and stemming and lowecasing done by analyzer module of lucene

Based on parsed data index will be created

Indexing in PyLucene

- Input: set of documents (corpus) to index;
- Output: index of documents.

```
import lucene

from org.apache.lucene.analysis.standard import StandardAnalyzer
from org.apache.lucene.document import Document, Field, StringField,
    FieldType
from org.apache.lucene.index import IndexWriter, IndexWriterConfig
from org.apache.lucene.store import SimpleFSDirectory, FSDirectory
```

* Basic packages of lucene required

Indexing in PyLucene

- Input: set of documents (corpus) to index; language;
- Output: index of documents.

```
import lucene

from org.apache.lucene.analysis.standard import StandardAnalyzer
from org.apache.lucene.document import Document, Field, StringField,
    FieldType
from org.apache.lucene.index import IndexWriter, IndexWriterConfig
from org.apache.lucene.store import SimpleFSDirectory, FSDirectory
```

Indexing in PyLucene

Preprocessing depends on ->

- Input: set of documents (corpus) to index; language; stopword list;
- Output: index of documents.

Stopwords depend on language

```
import lucene

from org.apache.lucene.analysis.standard import StandardAnalyzer
from org.apache.lucene.document import Document, Field, StringField,
    FieldType
from org.apache.lucene.index import IndexWriter, IndexWriterConfig
from org.apache.lucene.store import SimpleFSDirectory, FSDirectory
```

Starting with Pylucene

```
import lucene

lucene.initVM()
```

Initializing virtual machine
JCC -> C++ compiler -> Use java libraries from python wrapper -> we are interacting with VM by python

Tokenization using Pylucene

basic tokenizer example

```
from java.io import StringReader
from org.apache.lucene.analysis.standard import StandardTokenizer
from org.apache.lucene.analysis.tokenattributes import CharTermAttribute

test = "This is how we do it."
tokenizer = StandardTokenizer()
tokenizer.setReader(StringReader(test))
charTermAttrib = tokenizer.getAttribute(CharTermAttribute.class_)
tokenizer.reset()

tokens = []

while tokenizer.incrementToken():
    tokens.append(charTermAttrib.toString())
print(tokens)
```

Tokenization using Pylucene

standard tokenizer example

```
from java.io import StringReader
from org.apache.lucene.analysis.standard import StandardAnalyzer
from org.apache.lucene.analysis.tokenattributes import CharTermAttribute

analyzer = StandardAnalyzer()

stream = analyzer.tokenStream("", StringReader(test))
stream.reset()
tokens = []

while stream.incrementToken():
    tokens.append(stream.getAttribute(CharTermAttribute.class_).toString())
print(tokens)
```

Analysis using Pylucene

english analyzer example

```
from java.io import StringReader
from org.apache.lucene.analysis.standard import StandardAnalyzer
from org.apache.lucene.analysis.tokenattributes import CharTermAttribute

from org.apache.lucene.analysis.en import EnglishAnalyzer

test = "This is how we do it."
analyzer = EnglishAnalyzer()
stream = analyzer.tokenStream("", StringReader(test))
stream.reset()
tokens = []
while stream.incrementToken():
    tokens.append(stream.getAttribute(CharTermAttribute.class_).toString())

print(tokens)
```

Indexing in PyLucene

- Analyzer
- IndexWriter

Indexing in Lucene: Analyzer

```
# in general:
# import org.apache.lucene.analysis.ln.LangAnalyzer;
# analyzer = new LanguageAnalyzer();

# German analyzer
from org.apache.lucene.analysis.de import GermanAnalyzer
analyzer = new GermanAnalyzer();

# English analyzer
from org.apache.lucene.analysis.en import EnglishAnalyzer
analyzer = new EnglishAnalyzer();
```

Indexing in Lucene: IndexWriter

Hard disk or RAM where to store ?

Java

```
from org.apache.lucene.index import IndexWriter, IndexWriterConfig
from org.apache.lucene.store import SimpleFSDirectory, FSDirectory

indexPath = File("index/").toPath() #from java.io import File
indexDir = FSDirectory.open(indexPath) # RAMDirectory, DbDirectory,
    FileSwitchDirectory etc.

writerConfig = IndexWriterConfig(StandardAnalyzer())

writer = IndexWriter(indexDir, writerConfig)
```

package inside

RAM...

File System Directory

Variable

Path to store the index inside hard disk

If we provide a raw data to be indexed by default analyzing is done inside it

Make corresponding path specified earlier to be used for storing the index

Indexing in Lucene: Document Parser

- HTML
- XLS
- PDF
- CSV TSV
- TREC
- WARC
- ...

* We have to write/use based on data we will be manipulating.

-> CSV
-> Pandas -> Advantage -
Automatically header names will be
considered as indices of dataframe

Indexing in Lucene

parsing CSV files

```
movies = []
with open(file_path, newline='', encoding='utf-8') as f:
    reader = csv.reader(f, delimiter = ",", quotechar='"',
        quoting=csv.QUOTE_MINIMAL)
    c = 0
    for r in reader:
        c = c + 1
        print("Movie title {} - {}".format(c, r[1]))
        movies.append(r)
```

Indexing in Lucene: Index the Document

- For each document of the collection:

```
from org.apache.lucene import analysis, document, index, queryparser,
    search, store, util
import sys, os
from java.nio.file import Paths

def indexDocument(movie):
  fields[] = parseTSVformat(movie)
  doc = document.Document()
  doc.add(document.Field("title", fields[0], document.TextField.TYPE_STORED))
  doc.add(document.Field("year", fields[1], document.TextField.TYPE_STORED))
  doc.add(document.Field("plot", fields[6], document.TextField.TYPE_STORED))
  writer.addDocument(doc)
```

Lucene understandable document format

Indexing in Lucene: Index the Document

- For each document of the collection:

```
from org.apache.lucene import analysis, document, index, queryparser,
    search, store, util
import sys, os
from java.nio.file import Paths

def indexDocument(movie):
  fields[] = parseTSVformat(movie)
  doc = document.Document()
  doc.add(document.Field("title", fields[0], document.TextField.TYPE_STORED))
  doc.add(document.Field("year", fields[1], document.TextField.TYPE_STORED))
  doc.add(document.Field("plot", fields[6], document.TextField.TYPE_STORED))
  writer.addDocument(doc)
```

```
def closeIndexWriter():
  writer.close()
```