Apache Lucene

Apache lucene is a toolkit for indexing documents to easily perform search related tasks. It is written in java, so it runs on any operating system that the jvm can run on. It's main goal is to be a fast, and scalable search engine, while providing access to many different ranking algorithms.

For this tutorial we are going to go over some of the features of apache lucene uses the python wrapper, py lucene. We are going to cover how to index some documents, how to do searching using some different ranking algorithms, and some different preprocessing features that lucene offers.

For this tutorial we will be using Docker to manage all the installation. To run the container to get access to the container make sure docker and docker-compose are installed.

Code samples

All code can be found at https://github.com/colanconnon/cs410techreview

Setup

To install docker and docker compose follow these guides for your operating system.

- https://docs.docker.com/install/
- https://docs.docker.com/compose/install/

Once installed run `docker-compose run lucene bash`

To make sure everything is installed you can run `python test_lucene.py` you should see Lucene 7.5.0

Download data set

We will be using the imdb movie dataset

wget http://ai.stanford.edu/~amaas/data/sentiment/aclImdb_v1.tar.gz

sudo tar -zxvf aclImdb_v1.tar.gz

Using pylucene

All these scripts can be run in the docker file that is found in this repository.

Indexing

Before doing any searching lucene needs to index documents.

Indexing in lucene starts with creating documents. Documents have fields that can be anything from a title, to the contents of that document. Document fields have a type that controls how they are stored when indexed on disk.

Here is an example of creating a document with some field types.

```
from org.apache.lucene.document import Document, Field, FieldType

t1 = FieldType()
t1.setStored(True)
t1.setTokenized(False)
t1.setIndexOptions(IndexOptions.DOCS_AND_FREQS)

t2 = FieldType()
t2.setStored(False)
t2.setTokenized(True)
t2.setIndexOptions(IndexOptions.DOCS_AND_FREQS_AND_POSITIONS)

doc = Document()
doc.add(Field("name", filename, t1))
doc.add(Field("path", root, t1))
doc.add(Field("contents", contents, t2))
```

Field types are important because it controls how documents can be searched later. The link below shows all the options that can be set on a field type.

https://lucene.apache.org/core/7_5_0/core/index.html?org/apache/lucene/document/FieldType.html

Documents needs to be added to the index and written to disk.

We also use an analyzer, which takes care of preprocessing the text data. Analyzers can do things like tokenization, removing stop words, lower casing words, and other similar processing steps. Lucene has many different types of analyzers, and they can be used either when writing to an index or when searching.

```
store = SimpleFSDirectory(Paths.get(storeDir))
analyzer = LimitTokenCountAnalyzer(analyzer, 1048576)
config = IndexWriterConfig(analyzer)
config.setOpenMode(IndexWriterConfig.OpenMode.CREATE)
writer = IndexWriter(store, config)
writer.addDocument(doc)
```

For more reading about the types of analyzers that lucene offers:

https://lucene.apache.org/solr/guide/6_6/understanding-analyzers-tokenizers-and-filters.html
https://www.baeldung.com/lucene-analyzers

Custom analyzers can also be built.

https://github.com/svn2github/pylucene/blob/master/samples/PorterStemmerAnalyzer.py

Here is an example of building a custom analyzer in pylucene.

```
class PorterStemmerAnalyzer(PythonAnalyzer):

    def createComponents(self, fieldName):

        source = StandardTokenizer()
        filter = StandardFilter(source)
        filter = LowerCaseFilter(filter)
        filter = PorterStemFilter(filter)
        filter = StopFilter(filter, StopAnalyzer.ENGLISH_STOP_WORDS_SET)

        return self.TokenStreamComponents(source, filter)

    def initReader(self, fieldName, reader):
        return reader
```

You can use your custom analyzer like this

```
IndexFiles(STORE_DIR, INDEX_DIR, StandardAnalyzer())
```

A full script which will take a directory, and loop through all the files content and create documents that will be written to an index. This script was originally taken from https://github.com/fnp/pylucene/tree/master/samples and modified to index the imdb movies data set.

```
import sys, os, lucene, threading, time
from datetime import datetime

from java.nio.file import Paths
from org.apache.lucene.analysis.miscellaneous import LimitTokenCountAnalyzer
from org.apache.lucene.analysis.standard import StandardAnalyzer
from org.apache.lucene.document import Document, Field, FieldType
from org.apache.lucene.index import \
    FieldInfo, IndexWriter, IndexWriterConfig, IndexOptions
from org.apache.lucene.store import SimpleFSDirectory


class Ticker(object):

    def __init__(self):
        self.tick = True

    def run(self):
        while self.tick:
            sys.stdout.write('.')
            sys.stdout.flush()
```

```
            time.sleep(1.0)


class IndexFiles(object):

    def __init__(self, root, storeDir, analyzer):

        if not os.path.exists(storeDir):
            os.mkdir(storeDir)
        print(storeDir)
        store = SimpleFSDirectory(Paths.get(storeDir))
        analyzer = LimitTokenCountAnalyzer(analyzer, 1048576)
        config = IndexWriterConfig(analyzer)
        config.setOpenMode(IndexWriterConfig.OpenMode.CREATE)
        writer = IndexWriter(store, config)
        print(store)
        self.indexDocs(root, writer)
        ticker = Ticker()
        print 'commit index',
        threading.Thread(target=ticker.run).start()
        writer.commit()
        writer.close()
        ticker.tick = False
        print 'done'

    def indexDocs(self, root, writer):

        t1 = FieldType()
        t1.setStored(True)
        t1.setTokenized(False)
        t1.setIndexOptions(IndexOptions.DOCS_AND_FREQS)

        t2 = FieldType()
        t2.setStored(False)
        t2.setTokenized(True)
        t2.setIndexOptions(IndexOptions.DOCS_AND_FREQS_AND_POSITIONS)
        print(root)
        for root, dirnames, filenames in os.walk(root):
            print(filenames)
            for filename in filenames:
                if not filename.endswith('.txt'):
                    continue
                print "adding", filename
                try:
                    path = os.path.join(root, filename)
                    file = open(path)
                    contents = unicode(file.read(), 'iso-8859-1')
                    print(contents)
                    file.close()
                    doc = Document()
                    doc.add(Field("name", filename, t1))
                    doc.add(Field("path", root, t1))
                    if len(contents) > 0:
                        doc.add(Field("contents", contents, t2))
                    else:
                        print "warning: no content in %s" % filename
                    writer.addDocument(doc)
                except Exception, e:
```

```
                    print "Failed in indexDocs:", e

if __name__ == '__main__':
    lucene.initVM(vmargs=['-Djava.awt.headless=true'])
    print 'lucene', lucene.VERSION
    start = datetime.now()
    try:
        STORE_DIR = "/usr/src/pylucene/aclImdb/test/pos"
        INDEX_DIR = "/usr/src/pylucene/aclImdb/index"
        IndexFiles(STORE_DIR, INDEX_DIR,
                   StandardAnalyzer())
        end = datetime.now()
        print end - start
    except Exception, e:
        print "Failed: ", e
        raise e
```

Searching

Now that we have some documents indexed, we can start performing some search tasks. The searching is done on the index directory that was created by the indexer. In our code we need to find that directory, and pass it to the IndexSearcher. We also need an analyzer like we did when indexed the docs.

```
directory = SimpleFSDirectory(Paths.get(STORE_DIR))
searcher = IndexSearcher(DirectoryReader.open(directory))
analyzer = StandardAnalyzer()
```

We can also use different scoring functions when using the searcher. Here is an example of setting the bm25 scoring function to be used.

```
from org.apache.lucene.search.similarities import BM25Similarity
searcher.setSimilarity(BM25Similarity())
```

We can now use our analyzer and searcher to take a query, parse it and score some documents.

```
query = QueryParser("contents", analyzer).parse(command)
scoreDocs = searcher.search(query, 50).scoreDocs

for scoreDoc in scoreDocs:
    doc = searcher.doc(scoreDoc.doc)
    print 'path:', doc.get("path"), 'name:', doc.get("name")
```

Here is a full script for searching the index that was created earlier using.

```
import sys, os, lucene

from java.nio.file import Paths
```

```
from org.apache.lucene.store import SimpleFSDirectory
from org.apache.lucene.queryparser.classic import QueryParser
from org.apache.lucene.search import IndexSearcher
from org.apache.lucene.analysis.standard import StandardAnalyzer
from org.apache.lucene.index import DirectoryReader
from org.apache.lucene.search.similarities import BM25Similarity


def run(searcher, analyzer):
    while True:
        print
        print "Hit enter with no input to quit."
        command = raw_input("Query:")
        if command == '':
            return

        print
        print "Searching for:", command
        query = QueryParser("contents", analyzer).parse(command)
        scoreDocs = searcher.search(query, 50).scoreDocs
        print "%s total matching documents." % len(scoreDocs)

        for scoreDoc in scoreDocs:
            doc = searcher.doc(scoreDoc.doc)
            print 'path:', doc.get("path"), 'name:', doc.get("name")


if __name__ == '__main__':
    STORE_DIR = "/usr/src/pylucene/aclImdb/index"
    lucene.initVM(vmargs=['-Djava.awt.headless=true'])
    print 'lucene', lucene.VERSION
    directory = SimpleFSDirectory(Paths.get(STORE_DIR))
    searcher = IndexSearcher(DirectoryReader.open(directory))
    analyzer = StandardAnalyzer()
    searcher.setSimilarity(BM25Similarity())
    run(searcher, analyzer)
    del searcher
```

Conclusion

Apache Lucene is a very powerful toolkit that makes it very easy to build search capabilities very quickly. It has many prebuilt algorithms that let you easily experiment with what works best for your search problem, but it is also flexible enough that it can be extended to write custom algorithms. Many projects use lucene as a starting point for building more powerful searching technologies. Two examples of these are Apache Solr and Elasticsearch.

- http://lucene.apache.org/solr/
- https://www.elastic.co/

Sources:

http://lucene.apache.org/

http://lucene.apache.org/pylucene/index.html

https://en.wikipedia.org/wiki/Apache_Lucene

https://bitbucket.org/coady/docker/src/tip/pylucene/Dockerfile?fileviewer=file-view-default

https://github.com/fnp/pylucene/tree/master/samples

https://www.baeldung.com/lucene-analyzers