# Tech Review

MeTA || (Apache Lucene && Apache OpenLP) by Lowell Tyner

## Introduction

As we have learned in CS 410, some major strengths of the MeTA toolkit are that it combines both text search features (tokenization, indexing) and language processing features (POS tagging, grammatical parsing)[1]. This enables easier experimentation, making it ideal for learning and research. According to Professor Zhai, an author of MeTA, a motivation for this was the fractured nature of other tools for related areas. Since available toolkits evolve rapidly, the motivation behind this review is to see if MeTA still maintains this advantage, or whether similar toolkits can provide comparable benefits. Here we specifically focus on the combination Apache Lucene and Apache OpenNLP, comparing both functionality and ease of setup and use.

## Environment Details

Here we are comparing MeTA in Python (more specifically metapy) to the (native) Java libraries of Lucene and OpenNLP. For someone who is not familiar with Java, this may give an advantage to Python for ease of setup and use. However, this author is a Java developer who doesn't consider the boilerplate of Java code and project setup to be a roadblock. The system being used for this comparison is a Windows 11 machine with Python 3.6, Java 17 JDK, and Intellij for interacting with a Maven based Java project.

## Comparison Baseline

The functionality we will be comparing comes from CS 410's MP1, which covered topics such as metapy setup, tokenization, stopword removal, and POS tagging. With Python 3.6 installed, the metapy package installed, and the Python shell up and running, it was quite easy to follow the directions from MP1 to run all the commands in the Python shell. It's worth noting that Lucene has its own Python bindings, however this is not being explored here.

## Apache Lucene

Lucene is a "high-performance, full-featured text search engine library."[1] For this investigation we are using 8.10.1 version, and import the following dependencies in our Java project's pom.xml: "lucene-core", and "lucene-analyzers-common". The Javadoc[2] provides useful example code focused on indexing and searching. Using the example as inspiration, we populate a document with content from MP1:

```
Document doc = new Document();
String text = "I said that I can't believe that it only costs $19.95! I could
only find it for more than $30 before.";
doc.add(new Field("fieldname", text, TextField.TYPE_STORED));
```

It is of note that Lucene Documents consists of fields with text content, we can't directly set the content of the document like with MeTA. Now let's tokenize the same text:

```
StandardAnalyzer analyzer = new StandardAnalyzer();
TokenStream stream = analyzer.tokenStream("fieldname", doc.get("fieldname"));
stream.reset();
List result = new ArrayList<String>();
while (stream.incrementToken()) {
   result.add(stream.getAttribute(CharTermAttribute.class).toString());
}
stream.close();
System.out.println(Arrays.toString(result.toArray()));
```

Which outputs "[i, said, that, i, can't, believe, that, it, only, costs, 19.95, i, could, only, find, it, for, more, than, 30, before]." Note, we had to deal with clunky stream reset/close calls, and a clunky while loop/ boolean check to get all of the tokens out. Interestingly, the tokens are all lower case, even though the original text was not, unlike MeTA which required configuration to get lowercase. This is due to the StandardAnalyzer which uses opinionated but useful defaults.

In order to address stop words, we can create a StopFilter:

```
TokenStream filteredStream = new
StopFilter(stream,EnglishAnalyzer.ENGLISH_STOP_WORDS_SET);
```

With the same setup as above, this will output "[i, said, i, can't, believe, only, costs, 19.95, i, could, only, find, more, than, 30, before]." It is nice that there is a built in stop word set, although it only contains 33 words. Also, EnglishAnalyzer is in a separate package from lucene-core: "lucene-analyzers-common".

## Apache OpenNLP

OpenNLP is "a machine learning based toolkit for the processing of natural language text."[3] Here we are using the latest version 1.94 and have added the "opennlp-tools" dependency in our pom. In the following code we briefly explore part-of-speech tagging.

```
URL resource =
ClassLoader.getSystemClassLoader().getResource("en-pos-maxent.bin");
try(InputStream is = resource.openStream()) {
   POSModel model = new POSModel(is);
   String text = "The dog ran across the park.";
   POSTaggerME tagger = new POSTaggerME(model);
   String[] tokenized = WhitespaceTokenizer.INSTANCE.tokenize(text);
   String[] tagged = tagger.tag(tokenized);
   System.out.println(Arrays.toString(tokenized));
   System.out.println(Arrays.toString(tagged));
}
```

With the English maxent POS resource locally available, we get the following output, which matches the results from MeTA:

```
[The, dog, ran, across, the, park.]
[DT, NN, VBD, IN, DT, NN]
```

There is nothing particularly bulky or cumbersome about this chunk of code, with no major disadvantages compared to MeTA.

## Conclusion

Functionality-wise it is apparent that Lucene and OpenNLP combined are comparable to MeTA. Of course with a dedicated Java library, there will be layers of abstraction to sort through, and packages to import through trial and error.  This does allow for sensible default classes like StandardAnalyzer and built-in stop words, that can help avoid some configuration.  However, specifically Lucene's use of Java does not help its usability. I'd like to see the library make use of Java's Stream API to make concepts like tokens more accessible to the programmer in a fluent manner. Therefore the ease of accessing MeTA functionality from Python shell or simple script with simpler objects can't be beat when it comes to learning and exploring the concepts of text search and language processing.

## References

1. https://meta-toolkit.org/
2. https://lucene.apache.org/core/
3. https://lucene.apache.org/core/8_10_1/core/index.html
4. http://opennlp.apache.org/