COT6930 NLP
Dr Dingding Wang

Justin Johnson
Z23136514

Assignment 1
Part-of-Speech with Apache OpenNLP

Introduction)

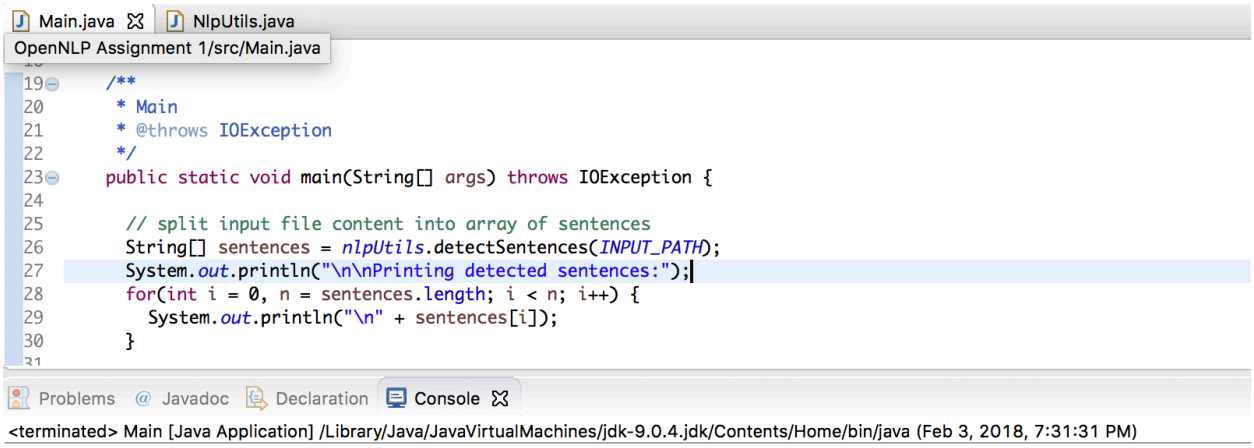
Sentence detection, tokenization, part-of-speech tagging, and name entity detection are performed on a one-page news article using Apache's OpenNLP library. OpenNLP is a machine learning toolkit for Java. OpenNLP also provides language specific pre-trained models that are used to complete several steps of this assignment. Each step is explained with screenshots of function definitions and results.

Part I) Sentence Detection

Sentence detection is achieved by reading a input file into a String, then passing the input String to an instance of OpenNLP's SentenceDetectorME class. The SentenceDetectorME object is instantiated with OpenNLP's pre-trained model, en-sent.bin.

```
50
51      /**
52       * Detect Sentences
53       * Uses OpenNLP pre-trained sentence detection model to construct array of sentences from input text
54       * @param inputFilePath path to input text
55       * @return String[] sentences
56       * @throws IOException
57       */
58      public String[] detectSentences(String inputFilePath) throws IOException {
59          String inputText = getStringFromFile(inputFilePath);
60          if(sentenceDetector == null) {
61              InputStream sentDetModelInput = new FileInputStream(SENT_DET_MODEL);
62              SentenceModel sentDetModel = new SentenceModel(sentDetModelInput);
63              sentenceDetector = new SentenceDetectorME(sentDetModel);
64          }
65          return sentenceDetector.sentDetect(inputText);
66      }
67
```

Below is a screenshot of the above function being invoked, and the corresponding results being printed to the console. The result is an array of sentences, as expected.



Part II) Sentence Tokenization

Next, each sentence is tokenized using OpenNLP's SimpleTokenizer. Provided a string, the tokenizeString function returns an array of tokens. The tokenization results will be displayed in Part III, with the POS results.

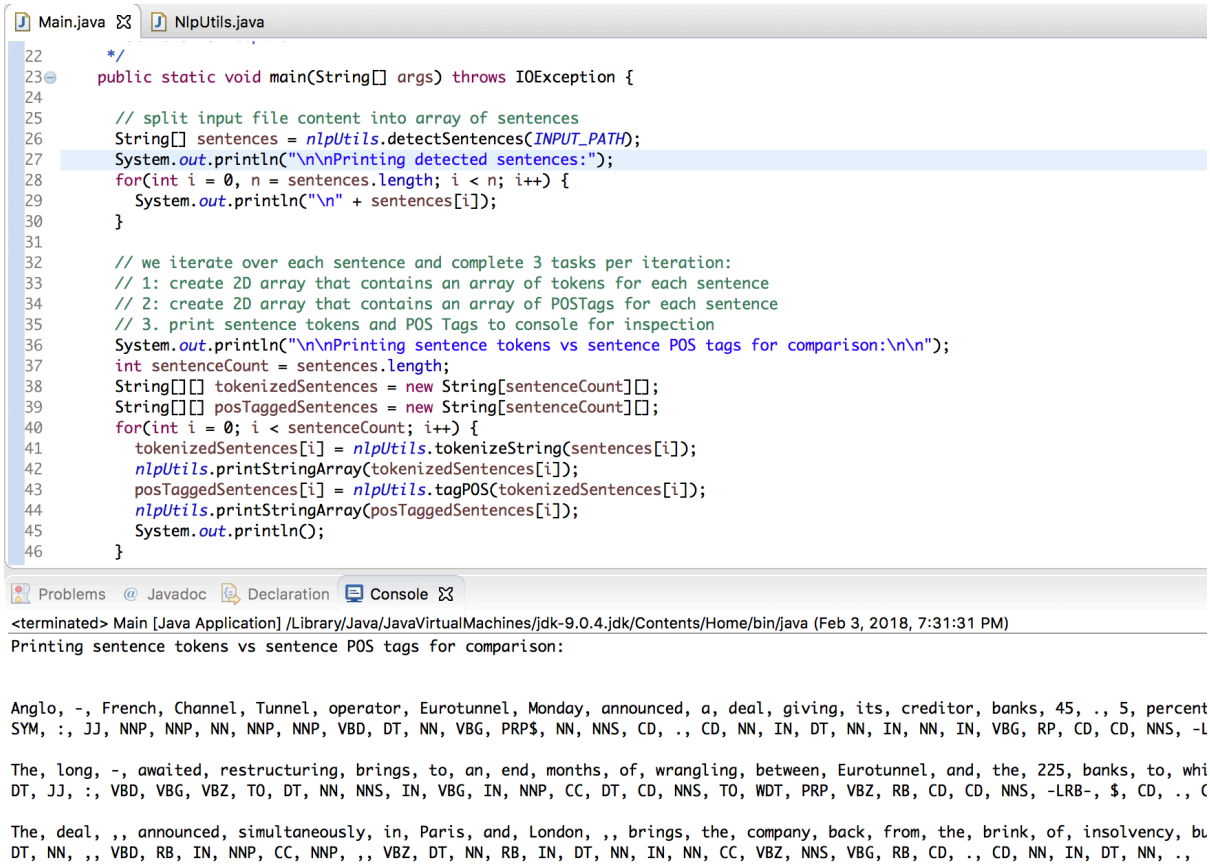
```
/**
 * Tokenize String
 * @param sentence
 * @return String[] of tokens
 */
public String[] tokenizeString(String s) {
    if(tokenizer == null) {
        tokenizer = SimpleTokenizer.INSTANCE;
    }
    return tokenizer.tokenize(s);
}
```

Part III) Part-of-Speech (POS) Tagging

Part-of-Speech tagging is completed using OpenNLP's POSTaggerME object. A POSTaggerME instance is initialized with OpenNLP's pre-trained model, en-pos-perceptron.bin. Given an array of tokens, the POSTagger returns an array of POS tags, such that the resulting array's i^{th} tag corresponds to the input token's i^{th} token.

```
/**
 * Tag tokens with part of speech (POS) tags
 * @param tokens to be tagged
 * @return String[] of POS Tags
 * @throws IOException
 */
public String[] tagPOS(String[] tokens) throws IOException {
    if(posTagger == null) {
        InputStream posTagModelInput = new FileInputStream(POS_TAG_MODEL);
        POSModel posTagModel = new POSModel(posTagModelInput);
        posTagger = new POSTaggerME(posTagModel);
    }
    return posTagger.tag(tokens);
}
```

The following screenshot displays tokenization and POS tagging in use, followed by the results. The array of sentences detected in Part I are traversed, creating an array of tokens for each sentence. The array of tokens is then passed to the POS tagging function, which then returns the POS tags that correspond to the array of tokens. Once completed for each sentence, the sentence's tokens are printed, and then the sentence's POS tags are printed.



Main.java | NlpUtils.java

```
22      */
23     public static void main(String[] args) throws IOException {
24
25       // split input file content into array of sentences
26       String[] sentences = nlpUtils.detectSentences(INPUT_PATH);
27       System.out.println("\n\nPrinting detected sentences:");
28       for(int i = 0, n = sentences.length; i < n; i++) {
29         System.out.println("\n" + sentences[i]);
30       }
31
32       // we iterate over each sentence and complete 3 tasks per iteration:
33       // 1: create 2D array that contains an array of tokens for each sentence
34       // 2: create 2D array that contains an array of POSTags for each sentence
35       // 3. print sentence tokens and POS Tags to console for inspection
36       System.out.println("\n\nPrinting sentence tokens vs sentence POS tags for comparison:\n\n");
37       int sentenceCount = sentences.length;
38       String[][] tokenizedSentences = new String[sentenceCount][];
39       String[][] posTaggedSentences = new String[sentenceCount][];
40       for(int i = 0; i < sentenceCount; i++) {
41         tokenizedSentences[i] = nlpUtils.tokenizeString(sentences[i]);
42         nlpUtils.printStringArray(tokenizedSentences[i]);
43         posTaggedSentences[i] = nlpUtils.tagPOS(tokenizedSentences[i]);
44         nlpUtils.printStringArray(posTaggedSentences[i]);
45         System.out.println();
46       }
```

Problems | Javadoc | Declaration | Console

```
<terminated> Main [Java Application] /Library/Java/JavaVirtualMachines/jdk-9.0.4.jdk/Contents/Home/bin/java (Feb 3, 2018, 7:31:31 PM)
Printing sentence tokens vs sentence POS tags for comparison:


Anglo, -, French, Channel, Tunnel, operator, Eurotunnel, Monday, announced, a, deal, giving, its, creditor, banks, 45, ., 5, percent
SYM, :, JJ, NNP, NNP, NN, NNP, NNP, VBD, DT, NN, VBG, PRP$, NN, NNS, CD, ., CD, NN, IN, DT, NN, IN, NN, IN, VBG, RP, CD, CD, NNS, -L

The, long, -, awaited, restructuring, brings, to, an, end, months, of, wrangling, between, Eurotunnel, and, the, 225, banks, to, whi
DT, JJ, :, VBD, VBG, VBZ, TO, DT, NN, NNS, IN, VBG, IN, NNP, CC, DT, CD, NNS, TO, WDT, PRP, VBZ, RB, CD, CD, NNS, -LRB-, $, CD, ., (

The, deal, ,, announced, simultaneously, in, Paris, and, London, ,, brings, the, company, back, from, the, brink, of, insolvency, bu
DT, NN, ,, VBD, RB, IN, NNP, CC, NNP, ,, VBZ, DT, NN, RB, IN, DT, NN, IN, NN, CC, VBZ, NNS, VBG, RB, CD, ., CD, NN, IN, DT, NN, .,
```

Part IV) Name Entity Detection

Finally, all sentences are checked for Name Entities (person and location). As displayed in below screenshot, OpenNLP's NameFinderME class is able to detect names for both people and locations. One instance of NameFinderME is given OpenNLP's pre-trained person entity model (en-ner-perons.bin) and the other is given a location entity model (en-ner-location.bin).

Main.java | NlpUtils.java

```
98      /**
99       * Detect Names
100      * @param nameType 'person' or 'location'
101      * @param tokens to be searched for names
102      * @return String[] of names found
103      * @throws IOException
104      */
105     public String[] detectNames(String nameType, String[] tokens) throws IOException {
106         if(nameType.equals("person")) {
107             Span[] spans = detectPersonNames(tokens);
108             return Span.spansToStrings(spans, tokens);
109         } else if(nameType.equals("location")) {
110             Span[] spans = detectLocationNames(tokens);
111             return Span.spansToStrings(spans, tokens);
112         } else {
113             throw new Error("Invalid name type: 'person' and 'location' are the two options");
114         }
115     }
116
117
118     /**
119      * Detect People Names
120      * @param tokens
121      * @return Span[]
122      * @throws IOException
123      */
124     private Span[] detectPersonNames(String[] tokens) throws IOException {
125         if(nameFinder == null) {
126             InputStream findNameModelInput = new FileInputStream(NAME_DET_MODEL);
127             TokenNameFinderModel findNameModel = new TokenNameFinderModel(findNameModelInput);
128             nameFinder = new NameFinderME(findNameModel);
129         }
130         return nameFinder.find(tokens);
131     }
132
133
134     /**
135      * Detect Location Names
136      * @param tokens
137      * @return Span[]
138      * @throws IOException
139      */
140     private Span[] detectLocationNames(String[] tokens) throws IOException {
141         if(locationFinder == null) {
142             InputStream findLocationModelInput = new FileInputStream(LOCATION_DET_MODEL);
143             TokenNameFinderModel findNameModel = new TokenNameFinderModel(findLocationModelInput);
144             locationFinder = new NameFinderME(findNameModel);
145         }
146         return locationFinder.find(tokens);
147     }
148
```

The models were able to identify 4 locations and 2 people. One word, Channel, was improperly identified as a location.

```
48
49        // next we will search each string for Name and Location entities
50        System.out.println("Starting Name and Location search...");
51        String[][] locations = new String[sentenceCount][];
52        String[][] people = new String[sentenceCount][];
53        for(int i = 0; i < sentenceCount; i++) {
54          locations[i] = nlpUtils.detectNames("location", tokenizedSentences[i]);
55          people[i] = nlpUtils.detectNames("person", tokenizedSentences[i]);
56        }
57
58
59        // print location results
60        System.out.println("\nLocations Found: ");
61        nlpUtils.printMatrixValues(locations);
62
63
64        // print people results
65        System.out.println("\nNames Found: ");
66        nlpUtils.printMatrixValues(people);
67
68      }
69
70  }
71
```

Problems @ Javadoc Declaration Console

<terminated> Main [Java Application] /Library/Java/JavaVirtualMachines/jdk-9.0.4.jdk/Contents/Home/bin/java (Feb

```
Starting Name and Location search...

Locations Found:
Paris, London, Channel, London,

Names Found:
Alastair Morton, Patrick Ponsolle,
```

Conclusion)

OpenNLP's natural language toolkit was successfully used to perform sentence detected, tokenization, part-of-speech tagging, and entity detection. Sentence detection, POS tagging, and entity detection utilized OpenNLP's pre-trained models. Tokenization was completed using OpenNLP's SimpleTokenizer.

The news-article.txt input file was then examined to evaluate the results of person and location entity detection. The name entity detection correctly returned 2 of 2 names from the news article, achieving 100% accuracy and 100% recall. The location entity detection selected 4 locations from the article, 1 of which (Channel) is a false positive. Location entity detection therefore obtained 75% precision and 100% recall.