Stanford CoreNLP - Introduction

19 December 2017
Svea Klaus
What it is

- set of human language technology tools
- Java annotation pipeline framework providing most of common core natural language processing steps:
  - Lower level functions such as tokenization
  - Higher level functions such as coreference resolution
- Supported languages: Arabic, Chinese, English, German, French

Manning et al. (2014): 55
Original use and Development

- initially designed for internal use → open source software since 2010
- coreNLP combines multiple language analysis components
- until 2006 each analysis component had their own ad hoc API
- now: uniform interface for annotators that add some kind of analysis information to some text
- Since 2009: command-line interface and ability to write annotations in various formats (e.g. XML)
- straightforward pipeline architecture
- simple, concrete API
- requires only very little Java knowledge
- easy to use and highly flexible + extensible
Elementary usage

- major goal: simple to setup and run processing pipelines (from either API or command-line)
- annotation pipeline can be applied to any text (paragraph, whole stories, etc.)
- minimal configuration code is required
System architecture

- annotator takes annotation object and adds additional information

Figure 1: Overall system architecture: Raw text is put into an Annotation object and then a sequence of Annotators add information in an analysis pipeline. The resulting Annotation, containing all the analysis information added by the Annotators, can be output in XML or plain text forms.
Example

www.corenlp.run/

or

http://nlp.stanford.edu:8080/corenlp/process
Advantages for the user

- fairly small and self-contained natural language analysis system, rather than a lot of functions the user won’t need anyway
- stable and robust high quality linguistic analysis components
- components are manageable with little NLP expertise
- project group attempts to answer user questions

Manning et al. (2014): 56
Annotators

- **tokenize:** tokenizes the text
- **ssplit:** splits tokens of text into sentences
- **lemma:** generates word lemmas for all tokens in the corpus
- **pos:** labels tokens with their part of speech tag
- **ner:** named entity recognition (person, company names etc.) → e.g. ner in English by default: named (person, location, organization), numerical (money, number, ordinal, percent), and temporal (date, time, duration) entities
- **parse:** gives full syntactic analysis (minimally a constituency (phrase structure tree) parse → if dependency parsing is available, it provides basic dependencies and/or enhanced dependencies)

Manning et al. (2014): 57
- **depparse**: fast syntactic dependency parser
- **coref**: coreference annotator, identifies mentions of the same entities in a text (for both nominal and pronominal coreference resolution)
- **natlog**: marks quantifier scope and token polarity (e.g. “all cats have tails” → ‘all’ is quantifier)
- **openie**: open information extraction, extracts open-domain relation triples - representing subject, a relation, and the object of relation (e.g. born-in (Barack Obama, Hawaii)
- **sentiment**: sentiment analysis (according to Socher et al.’s model: https://nlp.stanford.edu/sentiment/)
- **relation**: relation extraction, finds relations between two entities
- **quote**: picks out quotes from a text (considers “”, ′′, ` `, and ′′ as well as international quotation marks such as “”, ″, «», ‹›, 「」, 『』, and ′)
- **cleanxml**: removes XML tags from an input document
- **truecase**: recognizes the ‘true’ case of a token that might have been lost (e.g. capitalization in a well-edited text)
- **entitymentions**: generates list of mentions that have been identified by ner (named entity recognition)
- **udfeats**: labels tokens with universal dependencies, universal part of speech and features
Annotators in individual languages

<table>
<thead>
<tr>
<th>ANNOTATOR</th>
<th>AR</th>
<th>ZH</th>
<th>EN</th>
<th>FR</th>
<th>DE</th>
<th>ES</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tokenize / Segment</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Sentence Split</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Part of Speech</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Lemma</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>✓</td>
<td></td>
</tr>
<tr>
<td>Named Entities</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Constituency Parsing</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Dependency Parsing</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Sentiment Analysis</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>✓</td>
<td></td>
</tr>
<tr>
<td>Mention Detection</td>
<td>✓</td>
<td>✓</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Coreference</td>
<td>✓</td>
<td>✓</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Open IE</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>✓</td>
</tr>
</tbody>
</table>

Manning et al. (2014):59
## Annotator Dependencies

<table>
<thead>
<tr>
<th>Property Name</th>
<th>Annotator Class Name</th>
<th>Requirements</th>
</tr>
</thead>
<tbody>
<tr>
<td>tokenize</td>
<td>TokenizerAnnotator</td>
<td>None</td>
</tr>
<tr>
<td>cleanxml</td>
<td>CleanXmlAnnotator</td>
<td>tokenize</td>
</tr>
<tr>
<td>ssplit</td>
<td>WordsToSentenceAnnotator</td>
<td>tokenize</td>
</tr>
<tr>
<td>pos</td>
<td>POSTaggerAnnotator</td>
<td>tokenize, ssplit</td>
</tr>
<tr>
<td>lemma</td>
<td>MorphaAnnotator</td>
<td>tokenize, ssplit, pos</td>
</tr>
<tr>
<td>ner</td>
<td>NERClassifierCombiner</td>
<td>tokenize, ssplit, pos, lemma</td>
</tr>
<tr>
<td>regexner</td>
<td>RegexNERAnnotator</td>
<td>?</td>
</tr>
<tr>
<td>sentiment</td>
<td>SentimentAnnotator</td>
<td>?</td>
</tr>
<tr>
<td>parse</td>
<td>ParserAnnotator</td>
<td>tokenize, ssplit</td>
</tr>
<tr>
<td>depparse</td>
<td>DependencyParseAnnotator</td>
<td>tokenize, ssplit, pos</td>
</tr>
<tr>
<td>dcoref</td>
<td>DeterministicCorefAnnotator</td>
<td>tokenize, ssplit, pos, lemma, ner, parse</td>
</tr>
<tr>
<td>relation</td>
<td>RelationExtractorAnnotator</td>
<td>tokenize, ssplit, pos, lemma, ner, depparse</td>
</tr>
<tr>
<td>ntag</td>
<td>NaturalLogicAnnotator</td>
<td>tokenize, ssplit, pos, lemma, depparse (Can also use parse)</td>
</tr>
<tr>
<td>quote</td>
<td>QuoteAnnotator</td>
<td>None</td>
</tr>
</tbody>
</table>
Stanford CoreNLP - Installation | Basic Usage

19 December 2017
Madeeswaran Kannan
Prerequisites:

● SDK - Java 1.8
● IDE - Eclipse
● Maven (pre-installed with Eclipse)
Install SDK

Webpage -
(Java SE Development Kit 8u151)

Download the appropriate installer for your Operating System (x86 - 32-bit; x64 - 64-bit)

Run installer - Select “Development Tools” and “Public JRE”
Install IDE

Webpage -
https://www.eclipse.org/downloads/eclipse-packages/?osType=win32

Download the Eclipse installer for your Operating System

Run the installer - Select “Eclipse IDE for Java Developers”
Launch Eclipse - You'll see something similar to this (probably in a “Light” color scheme)
Install - CoreNLP

- Two methods: Manual installation vs Maven
  - Manual - Download libraries separately and add them to your Eclipse project’s Build Path as External JARs
    - Not recommended
    - Manual file management
    - Version conflicts
    - Dependency hell
Install - CoreNLP

- Maven - De-facto build tool for Java development
  - Manages project compilation and packaging
  - Automatically downloads libraries from a central repository
  - Support for versioning of libraries
  - Automatic dependency management
Create Maven Project

- File > New Project > Other... > Maven Project
- Maven template/archetype: Quickstart
- Group ID: Package name (e.g: com.nlptools)
- Artifact ID: Project name (e.g: corenlp)
Add CoreNLP Libraries

- Open pom.xml > Switch to "pom.xml" bottom tab
- Add the following inside <dependencies> and save:
  ```xml
  <dependency>
    <groupId>edu.stanford.nlp</groupId>
    <artifactId>stanford-corenlp</artifactId>
    <version>3.8.0</version>
  </dependency>
  <dependency>
    <groupId>edu.stanford.nlp</groupId>
    <artifactId>stanford-corenlp</artifactId>
    <version>3.8.0</version>
    <classifier>models-english</classifier>
  </dependency>
  ```
- Models for other languages can be added by changing "models-english" to "models-german", etc.
- Projects can use multiple language models at the same time.
Basic Usage
Stanford CoreNLP provides two APIs: Default & Simple

- **Default API**
  - Primary interface
  - Extensive & robust
  - Fully customisable

- **Simple API**
  - Newer implementation
  - Less customisable
Default API
Primary/Default API

Based on two classes: Annotation and Annotator

- **Annotation** class - Data structure that represents the results of Annotators
  - Key-Value Maps: Parse, NER, POS tags
- **Annotator** class - Functions that operate on Annotation objects
  - Encapsulate the core functionality
  - Perform tokenization, parsing, etc
  - Each (supported) NLP process is an Annotator
Primary/Default API

AnnotationPipeline class - Integrates Annotation and Annotator

- Sequence of annotators
- Example: Tokenize => Sentence Split => POS => Syntactic Parsing

StanfordCoreNLP class - Primary entrypoint
Primary/Default API

**Properties** class - Used to pass configuration data to pipelines and annotators

- Maps a property name (String) to its value (String)
- Values are always string representations of a given value type
  - Example: boolean values are represented as either “true” or “false”
Supports 18 annotators

Tokenization | Sentence Splitting | Lemmatization | Parts of Speech | Named Entity Recognition | RegEx
NER | Constituency Parsing | Dependency Parsing | Coreference Resolution | Natural Logic Extraction | Open Information Extraction | Sentiment Analysis | Relation Extraction | Quote Extraction | Clean XML | True Case Extraction | Entity Mentions
Using the primary API

1. Construct a `StanfordCoreNLP` object with some properties
   a. Properties should at least contain a list of annotators (list of annotator names)
      ```java
      Properties props = new Properties();
      props.setProperty("annotators", "tokenize, ssplit, pos, lemma, ner, parse, dcoref");
      ```
   b. Properties of annotators can be added in the following way: `<annotator_name>.<annotator_prop_name>`
      ```java
      Properties props = PropertiesUtils.asProperties(
          "annotators", "tokenize,ssplit,pos,lemma,parse,natlog",
          "ssplit.isOneSentence", "true",
          "parse.model", "edu/stanford/nlp/models/srparser/englishSR.ser.gz",
          "tokenize.language", "en")
      ```

      ```java
      StanfordCoreNLP pipeline = new StanfordCoreNLP(props);
      ```
Using the primary API

2. Create an `Annotation` object from the raw text
   ```java
   String text = "This is a simple sentence";
   Annotation document = new Annotation(text);
   ```

3. Run the annotators in the pipeline on the annotation
   ```java
   pipeline.annotate(document);
   ```

Annotators are run in the order they were declared in

- Some annotators may depend on others
Using the primary API

The output is accessed through the `CoreMap` and `CoreLabel` classes.

1. Use the `get()` method on the annotation object.

   ```java
   List<CoreMap> sentences = document.get(SentencesAnnotation.class);
   ```

   - Each annotation type has its own value type
   - A `CoreMap` is essentially a Map that uses class objects as keys and has values with custom types
   - A `CoreLabel` is a CoreMap with additional token-specific methods
Using the primary API

2. Traverse the words in the sentence and extract their annotation.

```java
List<CoreMap> sentences = document.get(SentencesAnnotation.class);

for(CoreMap sentence: sentences) {
    for (CoreLabel token: sentence.get(TokensAnnotation.class)) {
        // this is the text of the token
        String word = token.get(TextAnnotation.class);
        // this is the POS tag of the token
        String pos = token.get(PartOfSpeechAnnotation.class);
        // this is the NER label of the token
        String ne = token.get(NamedEntityTagAnnotation.class);
    }
}
```
Simple API
Simple API

Meant for use-cases that do not require a lot of customisation.

- Intuitive - Documents and Sentences are objects with functions that retrieve annotations from them
- Built on top of and yet hides the complexity of the primary API
- Syntax similar to other Python APIs
Simple API

Based on two classes: Document and Sentence

- **Document** - Represents a text containing multiple sentences
  - Wraps around multiple Sentence objects
- **Sentence** - Represents a single line of text
  - Tokens are array elements
  - Example: To get the lemma of a word, get the lemma array for the sentence and use the word's index
Using the simple API

1. Create a `Document` object from the raw text
   ```java
   Document doc = new Document("This is a document object! It can contain multiple sentences.");
   
   a. Lazy evaluation - no computation is done at this point

2. Iterate over the sentences and fetch annotations
   ```java
   for (Sentence sent : doc.sentences()) {
     // Will iterate over two sentences
     // When we ask for the lemma, it will load and run the part of speech tagger
     System.out.println("The third lemma of the sentence " + sent + "' is " + sent.lemma(2));
   }
   ```
Supports 10 annotators

Tokenization | Sentence Splitting | Lemmatization | Parts of Speech | Named Entity Recognition | Constituency Parsing | Dependency Parsing | Coreference Resolution | Natural Logic Extraction | Open Information Extraction
Java API Documentation -
https://nlp.stanford.edu/nlp/javadoc/javanlp/