Jigg

A Framework for an Easy Natural Language Processing Pipeline

Hiroshi Noji Nara Institute of Information Science and Technology (NAIST)
Yusuke Miyao National Institute of Informatics (NII)

https://github.com/mynlp/jigg or Google **“jigg nlp”**

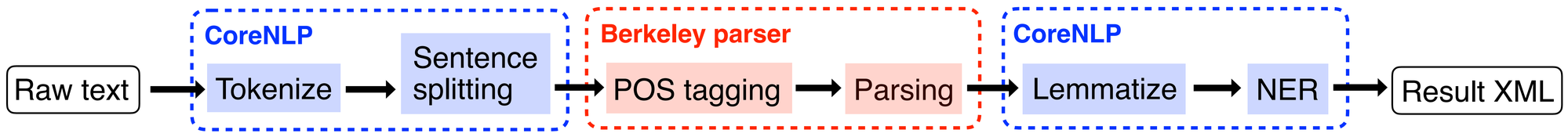
Quick start

No complex installation needed

```
$ wget https://github.com/mynlp/jigg/releases/download/v-0.6.1/jigg-0.6.1.tar.gz
$ tar xzf jigg-0.6.1.tar.gz && cd jigg-0.6.1 && ./script/download_corenlp_models.sh
```

Combining **Berkeley parser** with **Stanford CoreNLP pipeline** in one-line

```
$ echo “Jigg eats raw sentences. Like this.” | java -cp “*” jigg.pipeline.Pipeline \
  -annotators “corenlp[tokenize,ssplit],berkeleyparser,corenlp[lemma,ner]” > output.xml
```



Runs **Stanford NER** using **POS tags by Berkeley parser**

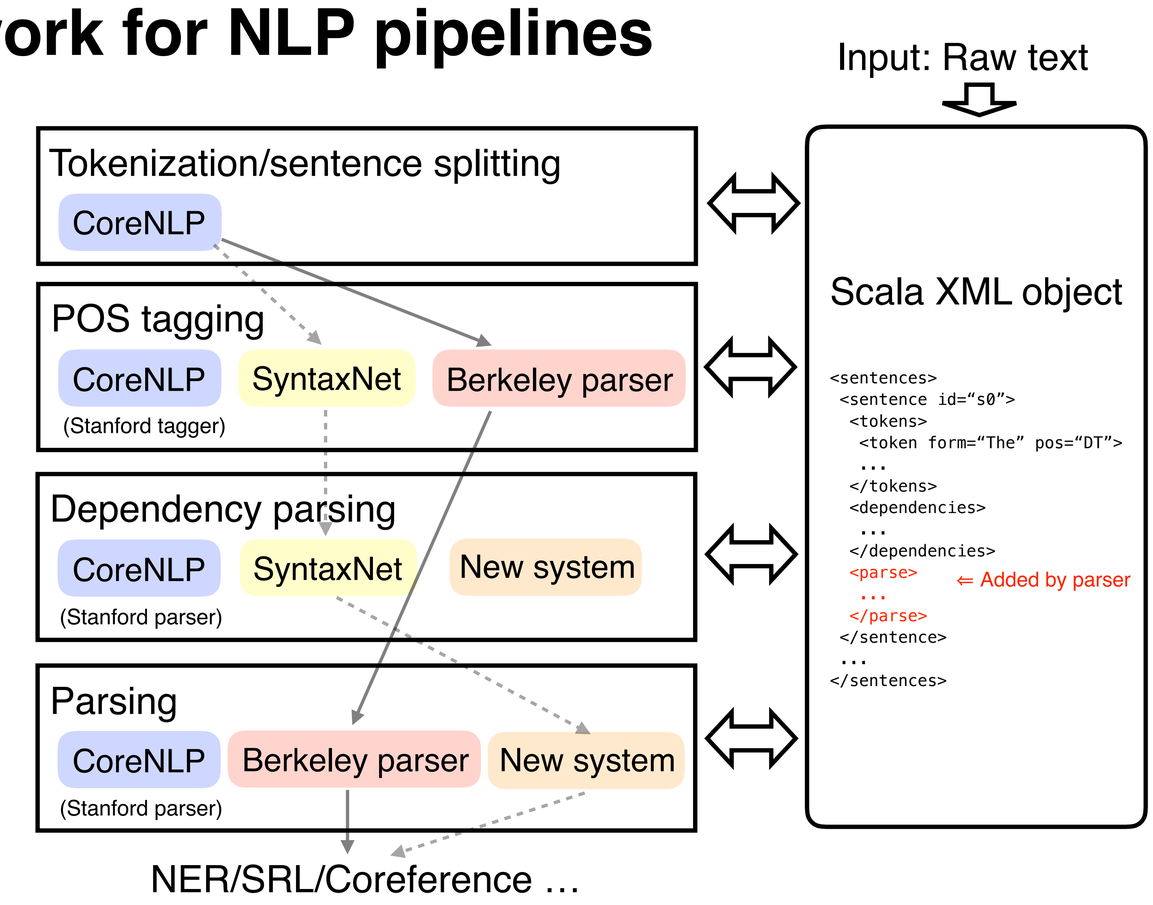
Jigg is a lightweight integrated framework for NLP pipelines

Motivations:

- Building NLP pipelines combining several tools is painful as the **input/output formats** often **vary** across tools
  - ex) CoreNLP ⇒ own XML; Google SyntaxNet ⇒ CoNLL format
- **Stanford CoreNLP** is a nice pipeline toolkit, but is less flexible; tools outside CoreNLP are not easily integrated into pipelines
- **Jigg** provides a flexible platform integrating various NLP tools

Features:

- (Almost) the same interface to Stanford CoreNLP
  - Customizable with Java properties file or command line; callable in Java
- Including CoreNLP itself in default (can be used transparently)
- **Easy to extend**: New pipeline component (**annotator**) can be added by writing a Scala/Java wrapper to the software
- Sentence/document-level **parallelization**: Most tools including Berkeley parser runs in parallel in default



- Each annotator wraps a system (software); annotates on Scala XML
- One can replace some component (e.g., parser) with a **new system**

Internal mechanism / Tips

Each annotator is implemented as a Scala (or Java) class:

```
package jigg.pipeline
import scala.xml._
class BerkeleyParserAnnotator extends SentenceAnnotator {
  val parser: CoarseToFineMaxRuleParser = ...
  override def newSentenceAnnotation(sentence: Node): Node = {
    // Use parser to get parse; add it to sentence XML (Node)
  }
  override def requires = Set(Tokenize, Ssplit)
  override def requirementsSatisfied = Set(POS, Parse)
}
```

These fields define dependencies between annotators (as in CoreNLP)

Correctness of the pipeline is checked before annotation:

- `-annotators “berkeleyparser,corenlp[lemma]”` will fail because `berkeleyparser` **requires** the input is already tokenized and ssplitted

You can extend Jigg by implementing new annotator class:

- If `jigg.pipeline.MSTParserAnnotator` is in the pass, this can be called by `-customAnnotatorClass.mst jigg.pipeline.MSTParserAnnotator`

If you distribute your annotator via maven, a third person can use it by customizing Jigg with build.sbt (or pom.xml):

```
libraryDependencies ++= Seq(
  "com.github.mynlp" % "jigg" % "0.6.1",
  "com.github.mynlp" % "jigg-mstparser" % "0.1-SNAPSHOT")
```

Discussion

- Jigg is inspired by CoreNLP in many aspects; the largest difference is in **annotated objects** (CoreMap in CoreNLP vs. Scala XML in Jigg)
- This design gives us more flexibility for the supportable annotators (e.g., supporting CCG parser in CoreNLP seems less obvious)
- Try Jigg! And give us feedback on **Github issue**!