# Hw1 NER System Using UIMA

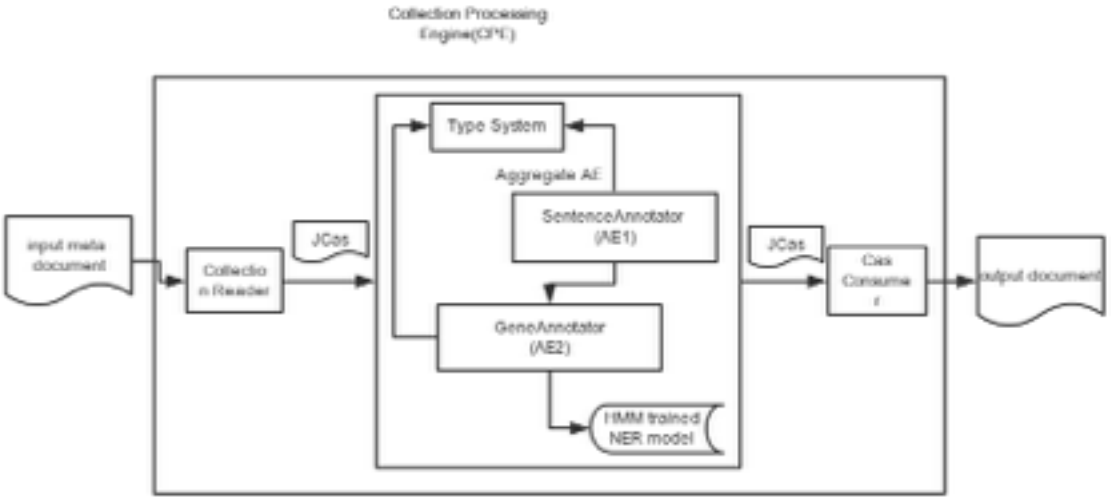
## 1 Overview

Basically, this report introduces the system design, and the core Statistical Named Entity recognizer of genes. I mainly refer to the First-Best Named Entity Chunking example in LingPipe tutorial. The model used by this chunker is ne-en-bio-genetag.HmmChunker, and it is included in the resources folder. Like other NER models, this one is labeled by task (ne for named-entity recognition), language (en for English), genre (bio for biology) and corpus (genetag for the GENETAG corpus), and suffixed with the name of the class of the serialized object (HmmChunker for com.aliasi.chunk.HmmChunker).

The following sections explain the details about this system, like system architecture, core techniques and algorithms, and performance evaluation based on the sample output.

## 2 System Design

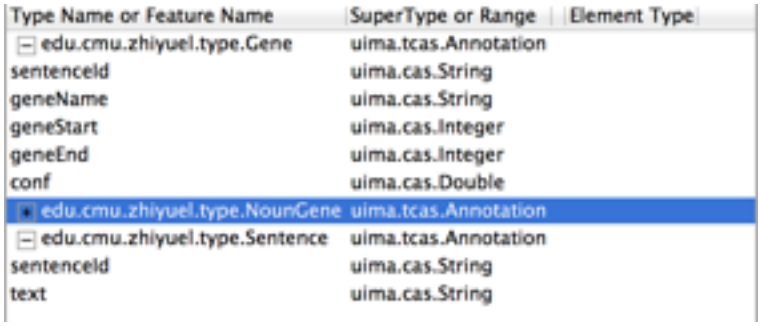
The system was built by UIMA pipeline, and its system flow is described in the following figure.




In UIMA Collection Reader handles the input, initialize the CAS. It reads the original document and converts into a format required by the Analysis Engine. In my system, the collection reader reads the file hw1.in and turn it into string *cas* and update the document text by using jcas.setDocumentText(cas).

The heart of this system is the Aggregate AE including several primitive analysis engines. All of them depends on the type system. In the type system, I have two types, that is sentence, and

Gene. Their attributes are illustrated in this picture. *SentenceId* is the span that identifies the sentence and is to connect both types. *Text* in Type Sentence stores the document context for each line. *GeneName* is to tag the names in language context. The attributes left is used for immediate output or further study. The type system generate java class for all the other class in the system.



| Type Name or Feature Name | SuperType or Range | Element Type |
|---|---|---|
| edu.cmu.zhiyuel.type.Gene | uima.tcas.Annotation | |
| sentenceId | uima.cas.String | |
| geneName | uima.cas.String | |
| geneStart | uima.cas.Integer | |
| geneEnd | uima.cas.Integer | |
| conf | uima.cas.Double | |
| edu.cmu.zhiyuel.type.NounGene | uima.tcas.Annotation | |
| edu.cmu.zhiyuel.type.Sentence | uima.tcas.Annotation | |
| sentenceId | uima.cas.String | |
| text | uima.cas.String | |

The SentenceAnnotator first populates the CAS generated by Collection Reader in the previous step. It also does the job of splitting up each line into two parts by using the first white space, one is for the sentenceId i.e. *P00001606T0076*, and the other is the sentence text like *Comparison with alkaline phosphatases and 5-nucleotidase*. After that, it again populates the CAS, to update the information of type Sentence in Jcas, that is to add the indexes for each sentence.

Another Analysis Engine is the GeneAnnotator which is the core component of my system. It extracts the sentenceId and sentence text from CAS. It utilizes the ne-en-bio-genetag.HmmChunker and identifies the geneTag from the sentence text. Also, it updates CAS, and add the indexes for each gene.

These two analysis engine compose the aggregate AE, and they follow fixed flow sequence, populates the CAS.

Finally, CAS consumer generates the results of CPE in a specific format which is set in its java file. It retrieves all the geneNames populated by AE, and writes to a pre-configured file.

# 3 Techniques, Algorithms, and Design Patterns

As the introduction of First-Best Named Entity Chunking example in LingPipe tutorial (showed in picture below), the algorithm basically, decides if an annotation is a named entity based on a Confidence Score generated by the model. One main concept is chunking, which is the identification of parts of speech and short phrases.

```
public static void main(String[] args) throws Exception {
    File modelFile = new File(args[0]);

    System.out.println("Reading chunker from file=" + modelFile);
    Chunker chunker
        = (Chunker) AbstractExternalizable.readObject(modelFile);

    for (int i = 1; i < args.length; ++i) {
        Chunking chunking = chunker.chunk(args[i]);
        System.out.println("Chunking=" + chunking);
    }
}
```

```
     [java] Chunking=p53 regulates human insulin-like growth factor II gene expression through active P4 promoter in rh
bdomyosarcoma cells. : [0-3:GENE@-Infinity, 20-54:GENE@-Infinity, 81-92:GENE@-Infinity]
```

Its outputs display a list of chunks found. It is said in the tutorial that the results show a chunk running from character position 0 (inclusive) to character position 3 (exclusive); another gene from position 20 to 54, and a final one from 81 to 92 for P4 promoter.

As for the design pattern, in this system, each class is assigned specific responsibilities using Information Expert Pattern. For example, CollectionReader is responsible for reading the input file. Another pattern is the creator pattern. Again, the CollectionReader initializes the CAS for class SentenceAnnotator.

# 4 Evaluation and Improvement

The evaluation scores based on *sample.out* in terms of Precision and Recall are given below.

- Precision : 0.7685139288192724
- Recall : 0.848836572679989

In fact, I first tried the PosTagNamedEntityRecognizer given by Stanford NLP, and it has a bad performance on recognizing the gene names. Many results of it is simple character like “r”, “q”. They are nouns, but not gene name. Another idea is to wrap a gene database into the system, and retrieve the nouns in the database. However, due to the limitation of gathering such data, I didn’t choose that approach.

I also tried to combine the Stanford NLP with this method using in my system. Results showed that there is no need to see if the geneName is a noun or not after using First-Best Named Entity Chunking, since the geneNames are already nouns. What’s more if using Stanford NLP after the method mentioned, it will reduce both precision and recall because it will split up the geneName into nouns that are not geneName. For example, some nouns like *factor* will be extracted. Details are written in *GeneNounAnnotator.java*.

By looking into the processing results, I found that there was too many noun phrases that did not actually correspond to a gene. Better results could be obtained if the chunking model is trained with a larger data set.

# 5 Reference

1.LingPipe tutorial
http://alias-i.com/lingpipe/demos/tutorial/ne/read-me.html
2. LingPipe demo
http://alias-i.com/lingpipe/web/demo-ne.html
3.uima example
4.deiis example given by piazza
5.UIMA Tutorial and Developers Guide
http://uima.apache.org/downloads/releaseDocs/2.1.0- incubating/docs/html/
tutorials_and_users_guides/tutorials_and_users_guides.html#ugr.tu g.aae.getting_started