11791: Design and Engineering for Intelligience Information Systems Homework 1 - Report

Name: Guanyu Wang, Andrew ID: guanyuw

Oct. 16th, 2012

In homework 1, I implemented three Name Entity Recognizers (NER) based on the **PosTag-NamedEntityRecognizer** (provided in the hw1-archetype), the gene annotation rules and the LingPipe tool kit[1]. In this report, I introduce the ideas of these three NERs as well as my idea of combination.

At first, I will give a general introduction of my understanding of the UIMA framework, Collection Processing Engine (CPE) combined with my own implementation.

1 Introduction - GenTag Type, CAS and CPE

1.1 Type System

Obviously, the NER which is required in this homework is an Analysis Engine Component, more specifically, an Annotator in the UIMA framework. Before I really start developing the Annotator, basically, the first step is to define the Common Analysis Structure(CAS) Feature Structure types for your own Annotator. This is done in an XML file called a Type System Descriptor. In this home work, my Annotator just need to recognize the Gene Mention, so it only need one type I call it **GenTag**, which is extended from the built-in type *uima.tcas.Annotation* [3], contains two more feature name: LineIndex(String) and mSofa(String). They are used to store the sentence-identifiers and the gene mentions recognized (cf. Figure 1).

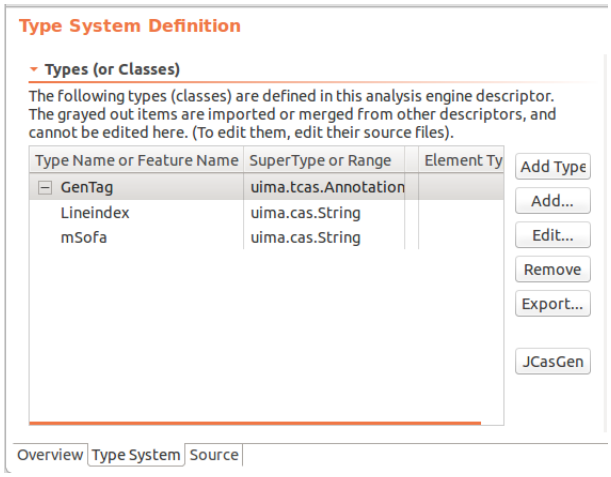

Figure 1: GenTag Type

1.2 Common Analysis Structure

Before go forward developing my NER, another thing is really important that I have to really understand in order to get into the main idea of the UIMA framework - the Common Analysis Structure (CAS).

CAS is the common data structure shared by all UIMA analytics to represent the unstructured information being analyzed (the artifact) as well as the metadata produced by the analysis workflow (the artifact metadata). The CAS is an Object Graph where Objects are instances of Classes and Classes are Types in a type system.

The above definition is given in [3]. Moreover, in my own understanding, the CAS provides a much easier way for developers to provides their own data without considering converting the format. All I need to do is just warping my meta data with a CAS "shell"' and offer it to the UIMA framework.

After defining the type for the Annotator, and understanding the CAS idea. I can start coding my own NER now, which is described precisely in the Section 2.

1.3 Collection Processing Engine

The last thing that I have to mention in this introduction section is the CPE. Even I have implemented a well-done NER, it is hard to test the outputs (e.g. precision, recall, etc.) from it. (Actually I can still test my own Annotators with the UIMA SDK's tools, like Document Analyzer, GUI CPE, etc. But still, they are not that convenient.) So really implementing a CPE with your own hand is important to under UIMA as well as the work flow in it.

The CPE mainly contains three components [2]: Collection Reader, Analysis Engine and

CAS Consumer, cf. Figure 2.

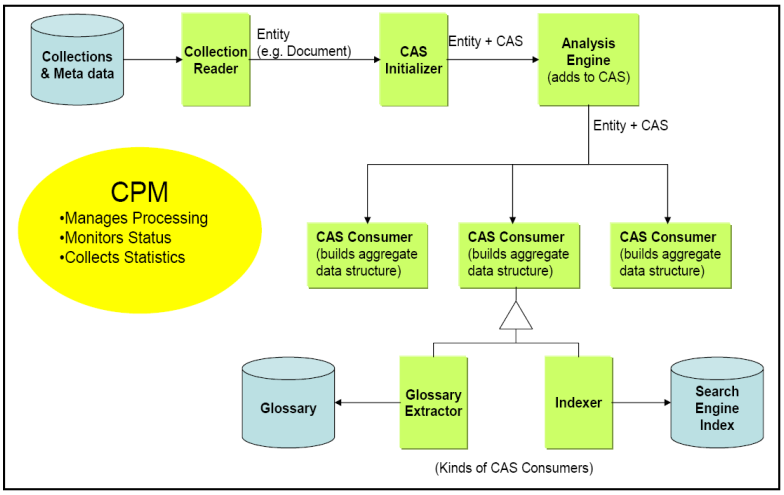

Figure 2: CPE components in [2]

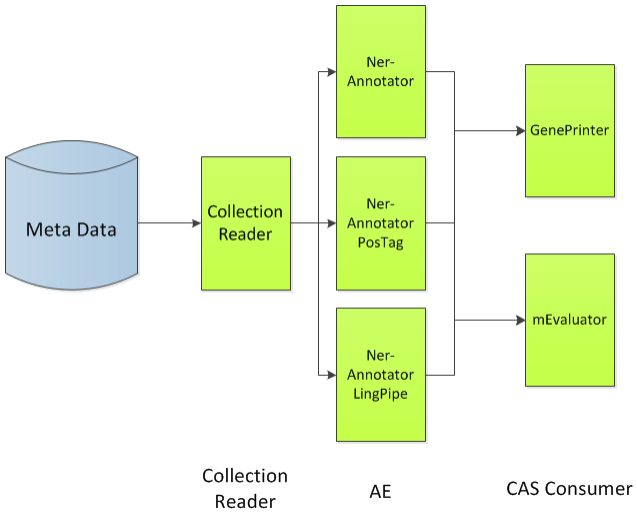

Figure 3: My CPE components

In my implementation, there is one Collection Reader which is used to read the input file which is ready to do the name entity recognizing. I have three Analysis Engines, i.e. NERs, in my system, **NErAnnotatorPosTag**, **NErAnnotatorLingPipe**, and **NErAnnotator** (I this NER's output as my final output), the first one is based on the **PosTagNamedEntityRecognizer** given in the hw1-archetype as well as the gene annotation rules given in the write-up. The second one is based on the well-known LingPipe tool kit. The last one is a combination of previous two tagging ideas. Also, I have two CAS Consumers. One named **GenePrinter** is used to output the final reported gene name mentions and the other

one is used to task the final tagged result (comparison with the given sample.out file), cf. Figure 3.

So briefly, the general data flow in my system is: **CollectionReader** read all sentences from the input file (default as sample.in), then there is one of the three AEs (**NErAnnotator**, **NErAnnotatorPosTag** or **NErAnnotatorLingPipe**) to process all the sentences and package them into CASs (JCas in Java) with specified annotations. Finally, I can choose to use **GenePrinter** to output a file with all tagged gene mentions in the required format, or to use **mEvaluator** to output a test report including count number of different tags, precision, recall (based on the sample.out).

2 My Name Entity Recognizers

2.1 NErAnnotatorPosTag

This NER is mainly based on two components: **PosTagNamedEntity** Class and **GeneRuler** Class. As my first attempt, I adopt the **PosTagNamedEntity** given in the hw1-archetype only. According to the evaluation, the precision is very low, about 10%. But at the same time, the recall is about 55%, which means it accepts too many wrong answers even though most of the right gene mentions can also be tagged.

In order to increase the precision, I have to use more strict constraints to filter the wrong answers. The **GeneRuler** Class is constructed with this purpose. According to the GENE-TAG Annotation Guidelines, there are several patterns which implies the given word is a gene mention. Moreover, there are also several "forbidden" types. The following are four pre-defined patterns used in my **GeneRuler**:

- MatchString: Many key words which appear a lot in the gene mentions, like enhancer, receptor, etc.
- Qualifiers: Many suffixes which appear a lot in the gene mentions, like alpha, I, 1, etc.
- Hormones: The names and abbreviations of Peptide hormones.
- notAllow: The patterns which do not appear in the gene mentions.

First, if the MatchString is a substring of the given text, it has a chance to be a gene mention. Second, if the given text has Qualifiers as suffix, it has a chance to be tagged as gene mention. Third, all the Peptide hormones in Hormones should be tagged as gene mentions. Finally, All the gene mention candidates should be purified with these forbidden patterns in notAllow.

So the basic idea is that the original text can be tokenized (and basically filtered), then all these tokens selected should be processed by the **GeneRuler**. I show the result comparison

between the tagging using just "**PosTagNamedEntity**" and "**PosTagNamedEntity + GeneRuler**".

	PosTagNamedEntity	**PosTagNamedEntity + GeneRuler**
Precision	0.10254172	0.29433373
Recall	0.54667395	0.26761565
Total Time	21871ms	19878ms

Table 1: Evaluation of output from **NErAnnotatorPosTag**

2.2 NErAnnotatorLingPipe

Another idea to design a NER is the machine learning mechanism. There are many ideas can be considered, like Neutral Network, Conditional Random Field and hidden Markov model, etc.

The **NErAnnotator** outputs better result compared with the first one (described in Section 2.1). Basically I adopt the *HmmChunker* in LingPipe tool kit to do the tokenizing and tagging tasks. It uses a hidden Markov model to perform chunking over tokenized character sequences. Any instances of the HmmChunker Class contain a hidden Markov model, decoder for the model and tokenizer factory.

About the hidden Markov model, it is a well-known statistical model which is very similar with the common Markov model. The difference is that its state is not directly visible. But the visible output dependents on the states, so using the outputs, the model can infer the hidden states, which provides a typical training model. Here, since the given sample.in file's data amount is limited, the training result is hard to use. So I choose the well-trained model from LingPipe tool kit.

The evaluation of result for sample.in is:

Precision	0.7685139
Recall	0.8488366
Total Time	3712ms

Table 2: Evaluation of output from **NErAnnotatorLingPipe**

2.3 NErAnnotator

This NER is a combination of **NErAnnotatorLingPipe** and **NErAnnotatorPosTag**, which can be viewed as a aggregated NER. The main ideas is that the **NErAnnotatorLingPipe** and **NErAnnotatorPosTag** have very different tagging methods, so they probably

have very different tagging result. Then collecting all the outputs from two NERs may give a high recall. The later evaluation result also verifies this assumption.

The important function for the **NErAnnotator** is aggregation. Here I adopt the easiest method: discard the duplicate CASs. Basically, this NER deals with the whole given input line by line. For any line, it compares the result of **NErAnnotatorLingPipe** and result of **NErAnnotatorPosTag** based on the membership testing approach: store all the tag of one line from **NErAnnotatorLingPipe** in a HashSet, and test the containing property with the output of the same line from **NErAnnotatorPosTag**.

The evaluation of result for sample.in is:

Precision	0.50735176
Recall	0.8633452
Total Time	35331ms

Table 3: Evaluation of output from **NErAnnotator**

Here, I failed to find some duplicate-removal methods for the CAS of UIMA. Maybe there is one and I just cannot find it. If not, I think a very efficient approach to remove all the duplicate CASs from the indices can provide many advantages for not only analysis aggregation but also CAS consumer design.

3 Conclusion

According to the evaluation results: the **NErAnnotatorLingPipe** gives the best precision and processing time. But the **NErAnnotator** provides a acceptable precision (in the middle of **NErAnnotatorLingPipe** and **NErAnnotatorPosTag**) and the best recall. Of course, it has one shortcoming: it costs about 30s to process the sample.in file, which is longer than **NErAnnotatorLingPipe**. So **NErAnnotatorLingPipe** and **NErAnnotator** have their own advantages. I adopt the **NErAnnotator** as my final NER in the Cpedescriptor file.

References

[1] LingPipe Homepage. `http://alias-i.com/lingpipe/index.html`. [Online; accessed 14-Oct-2012].

[2] UIMA Tutorial and Developers' Guides. `http://uima.apache.org/d/uimaj-2.4.0/tutorials_and_users_guides.html`. [Online; accessed 14-Oct-2012].

[3] David Ferrucci, Adam Lally, Karin Verspoor, and Eric Nyberg. Unstructured Information Management Architecture (UIMA) Version 1.0, OASIS Standard. Technical report, OASIS, March 2009.