F12-Software Engineering for Information Systems
Assignment 1

Ting-Hao (Kenneth) Huang

- **Introduction**

  In this assignment, we're required to tackle the Gene Mention Tagging task, which is the named entity extraction task focusing on the gene and gene product. The TA provided a simple code which just uses Stanford CoreNLP tool to annotate all consecutive nouns as named entities. Our goal is to build a robust pipeline by UIMA and try to achieve better performance.

- **System Design**

  Under the UIMA framework, the better approach for this task is to process each sentence as a single JCas rather than process the whole document as one giant JCas. This approach has two apparent advantages: First, even the input file is abnormally huge, e.g., 4G or more, it can still be handled without exceeding the system memory. Second, the final output format requires the string offsets which are relative to the start of the sentence, but not relative to the whole document. By putting only one sentence in each JCas, we can save some extra effort on maintaining the offsets among multiple sentences.

  In this assignment, we plan to build a 3-component pipeline. The first component is the collection reader *InputFileCollectionReader*, which takes the input file, does the enumeration on all lines, generates JCas for each line, and annotates the Sentence label; the second component is the analysis engine *NamedEntityAnnotator*, which takes the JCas and annotate the NamedEntity labels; and the final one is the Cas consumer *OutputWriter*, which takes the resulting jCas and output to the file.

- **Type System**

  In our type system, we only have two types: "NamedEntity" and "Sentence." The "NamedEntity" type has 3 features: *neType* for named entity type, *sentId* for sentence ID, and *text*. The "Sentence" type also has 2 features: *sentId* and *text*.

Our goal is to make our type system as simple as possible. So we don't really associate NamedEntity object with Sentence object, but only keep the *sentId* in the NamedEntity type.

Note that in our system design, the "Sentence" type is actually not necessary. Because each JCas contains only single sentence and the sentences are already segmented from the input. However, we still keep the "Sentence" type in our type system because we were concerning the situation that some lines would accidently contain more than one sentence. Though it's very likely not to happen and even it happens perhaps none of our components would be able to recognize it, considering the fact that we adopted two external tools in this assignment and we don't really understand all internal details of those tools, we finally decide to keep the "Sentence" type in our type system.

- **Techniques and External Tools**

In this assignment, we mainly adopted these two external tools to recognize named entities:

- LingPipe Named Entity Recognizer[1]

  LingPipe toolkit proves a named entity recognizer which is based on HMM and trained for the English GeneTag task.

- JULIE Lab Named Entity Tagger (JNET)[2]

  JULIE Lab Toolkit also provides a named entity tagger for GeneTag task. Though this tool comes with the UIMA analysis engine form (in PEAR file format), we still choose to hack the source java code to fit our type system.

In our system, we first obtained the named entity labels from these two tools, and then remove the duplicate labels when outputting. We'll describe the details in the following section.

- **Issues**

[1] http://alias-i.com/lingpipe/demos/tutorial/ne/read-me.html
[2] http://www.julielab.de/Resources/Software/NLP_Tools.html

- Offset Maintenance

  In this assignment, the output format requires the offsets of spans which are relative to the start of the sentence, and the offsets should ignore all the white space. Therefore, the maintenance of string offsets becomes an issue.

  Our solution is to keep every string offset as the original Java offset in our JCas, only modify it into the required offset when outputting. The advantage of this approach is that it clearly cause less confusion in development, and easier to integrate the different outputs from different third-party software.

- Token Issues of the Julie Lab Toolkit

  We found that the Julie Lab NER tool has some problems if we feed the tokens obtained from the Stanford CoreNLP tools. We traced the code of Julie Lab Toolkit and found that these problems are caused by the different token offset (including begin and end index) calculation methods between two different tools. It's too expensive if we have to hack the code deeply to solve it. So instead, we just do the naïve tokenization by using the white space and feed the resulting tokens to the Julie Lab NER tool.

- Output Combination

  The method we adopted to combine the outputs from two different named entity recognizers is just removing the same spans, i.e., the spans with exactly the same begin and end indexes.

  Though we understand that this approach would create some noise in our output, e.g., some overlap of recognized spans, considering the evaluation might requires exactly the same begin and end to match, we feel like keeping the overlap spans is better than to merge them or to remove them all.

- **Result**

We ran some experiments to test the performance of our system. As a result, LingPipe recognizer achieves 0.80 in F1-score, and if combined with Julie Lab Tool, the F1-score decreases to 0.78, but has slightly higher recall value.