11-791 Design and Engineering of Intelligent Information System Fall 2012

Homework 1 Report

Yuan Gu

yuangu@andrew.cmu.edu

1, Overview

1.1 Project Overview

In this project, a named entity recognition system is developed with UIMA SDK. As is required, the input of the system is a file with multiple lines and each line represents a sentence with the following format:

sentence-identifier-1 text

The developed system could recognize named entities in the text and output in the following format:

sentence-identifier-1|start-offset-1 end-offset-1|optional text...

Based on the framework of UIMA SDK, our work is divided into the following part:

- Type System Design: design a type system with two types, one of which servers as annotation type to record information about each annotation made in the system, the other servers to record some meta information about the original sentence.
- Collection Reader Design and Development: design and develop a collection reader could parse the input file and convert each sentence into CAS for successive processing.
- Analysis Engine Design and Development: design and develop three analysis engines (annotators) to process the CAS generated by Collection Reader, recognize the named entities in the sentence and add the recognized named entities as annotations into the CAS.
- CAS Consumer Design and Development: design and develop two CAS consumers, one of which could output the CAS with the required format, the other one could evaluation the performance of the whole system analysis.
- Experiments: conduct some experiments on different configurations of the system.

In the following chapters, we will give detailed descriptions about each components in the system and the experiments we conducted.

2, Type System

Our type system has two types:

- BaseAnnotation: The BaseAnnotation type is the basic annotation type in the system. It inherits *uima.tcas.Annotation*, with two additional fields added:
  - source (uima.cas.String): as our system have more than one analysis engines, the source field is used to record the analysis engine who makes the corresponding annotation.
  - confidence (uima.cas.Float): as some the engines make annotation by probability models, the confidence field is used to record the confidence the annotator makes the corresponding annotation.
- SourceSentenceInformation: The SourceSentenceInformation type is used by the collection reader to record the meta information about a sentence, like the sentence identifier in the input file. The SourceSentenceInformation type also inherits *uima.tcas.Annotation*, with one additional field added:
  - identifier (uima.cas.String): the identifier field is used to record the sentence's identifier in the file.

3, Collection Reader

Collection reader needs to process the input of the system and produces CAS objects. As the focus of our system is to process sentence, we define a collection reader named *LineBreakCollectionReader*, which breaks the input file into lines, and for each line it creates a CAS. Successive components in the system will each time process a CAS.

LineBreakCollectionReader could be configured with the following parameter:

- *InputFile* – path to the input file

4, Analysis Engine

Analysis engines are the key components in our system. They are responsible to add annotations for each CAS generated by the *LineBreakCollectionRead*. We have three

analysis engines in the system:

- *GeneRegexAnnotator*: This is a simple analysis engine, which makes annotations by parsing the sentence with some hand-written regular expressions. This annotator is mainly used to test the system's integrity.
- *GeneStanfordNLPAnnotator*: This analysis engine wraps the Stanford NLP NE recognizer.
- *GeneLingPipeAnnotator*: This analysis engine is based on the LingPipe library. It utilizes the LingPipe named entity recognition module.
- *GeneMentionFilter*: This analysis engine is responsible to filter out low quality annotations made by other analysis engines.

Here we give a detailed description of these three analysis engines.

4.1 GeneRegexAnnotator

GeneRegexAnnotator is very simple. During initialization, it reads in the regular expressions as parameters and each time the UIMA framework to process a CAS calls it, it simply iterates over all regular expressions and output the matched parts as annotations.

GeneRegexAnnotator could be configured with the following parameter:

- *GeneRegularExpressions* – regular expression strings to match annotations

4.2 GeneStanfordNLPAnnotator

While designing the *GeneStanfordNLPAnnotator*, we utilize the adaptor design pattern: every *GeneStanfordNLPAnnotator* instance contains an instance of the class *PosTagNamedEntityRecognizer* provided as an example in the assignment. Thus, each time *GeneStanfordNLPAnnotator* processes a CAS, it calls the *getGeneSpans()* method of *PosTagNameEntityRecognizer* and get a list of named entity annotations. It then converts these annotations into the type BaseAnnotation and adds them to the CAS's indexes.

GeneStanfordNLPAnnotator has no configurable parameter.

4.3 GeneLingPipeAnnotator

GeneLingPipeAnnotator is another example of the adapter design pattern. It wraps the *chunker* class provided by the named entity recognition module of the LingPipe library. Each time it is called by the UIMA framework to process a CAS, it calls the *chunker* class's *chunk()* function to get a set of *chunks*. It then iterates over these chunks and converts them into BaseAnnotation instances and adds to the CAS's indexes.

As the named entity recognition module of the LingPipe library utilizes a statistical model, we need to provide it a trained model. To avoid over fitting, instead of training a new model on the sample provided in the assignment, our system uses a model provided by the LingPipe library. *GeneLingPipeAnnotator* uses the UIMA resource management system to manage the model file.

GeneLingPipeAnnotator could be configured with the following parameter:

- *BestChunkNumber* – number of best chunks when annotating with LingPipe confidence chunker

4.4 GeneMentionFilter

GeneMentionFilter is responsible to examine the annotations output be the previous processing engines and filter out low quality annotations. Currently, the *GeneMentionFilter* will filter out all annotations whose confidences are lower than a threshold, which is configured as a parameter of *GeneMentionFilter*. It will also filter out annotations matched with some regular expressions, which are passed as parameter to *GeneMentionFilter*.

GeneMentionFilter could be configured with the following parameter:

- *ConfidenThreshold* – confidence threshold to filter annotations, annotations with lower confidence than this value will be filtered out.
- *FilteringRegularExpressions* – regular expressions to filter annotations, annotations matched with these expressions will be filtered out.

5, CAS Consumer

Our system has two types of CAS consumers:

- *AnnotationPrinter*: *AnnotationPrinter* is responsible to output the annotations to a file, as is required by the assignment.
- *AnnotationEvaluator*: As the assignment provides a sample for the input and output file, which could server as a testing case for our system, we design and develop the *AnnotationEvaluator* consumer. It is responsible to evaluate the system's performance (recall and precision) by comparing the system's output with the sample.out.

Here is the detailed description of these two CAS consumers.

5.1 AnnotationPrinter

AnnotationPrinter takes one parameter that specifies where to output the results. While processing a CAS, *AnnotationPrinter* firstly finds the annotation with the type *SourceSentenceInformation* and extracts the meta information about the sentence. Then it iterates over the annotations whose type is BaseAnnotation and output the annotation. As the assignment requires output non-space version of each named entity's offset, and the analysis engines generate annotations based on the true offset in the sentence (with spaces), *AnnotationPrinter* will also do some conversion to get the non-space version of beg and end positions before it outputs each annotation.

AnnotationPrinter could be configured with the following parameter:

- *OutputFile* – path to the output file

5.2 AnnotationEvaluator

AnnotationEvaluator also takes one parameter, which specifies where the true answer file is. Then at initialization, *AnnotationEvaluator* will read in all the true answer annotations into memory. Then it does a similar job to *AnnotationPrinter*: iterating over each BaseAnnotation and converts them into results in output format. However, *AnnotationEvaluator* will not output the results to a file. Instead, it will compare the results towards true answers to check whether they could match. After all, when the processing is complete, it will print some performance metrics on the

console.

AnnotationEvaluator could be configured with the following parameter:

- *TrueAnswerFile* – path to the true answer file

6, Performance Tuning and Evaluation

To see how different parameters and the combination of analysis engines would affect the system's performance, we conduct several experiments on the system.with different configurations.

The experiments are conducted with the following configurations:

- System Input: sample.in provided in the assignment
- System Output: output generated by the system
- True Answer: sample.out provided in the assignment

The result is shown below[1]:

AE Combination	Parameter Configuration	Recall	Precision	F-measure
StanfordNLP	N/A	0.55	0.10	0.17
LingPipe	*BCN* = 5	0.89	0.19	0.31
LingPipe	*BCN* = 10	0.97	0.11	0.20
LingPipe	*BCN* = 15	0.99	0.08	0.14
LingPipe	*BCN* = 20	0.99	0.06	0.11
LingPipe + Filter	*BCN* = 5, ConfThr = 0.5	0.80	0.80	0.80
LingPipe + Filter	*BCN* = 10, ConfThr = 0.5	0.84	0.79	0.81
LingPipe + Filter	*BCN* = 15, ConfThr = 0.5	0.84	0.79	0.81
LingPipe + Filter	*BCN* = 20, ConfThr = 0.5	0.84	0.79	0.81
LingPipe + Filter	*BCN* = 10, ConfThr = 0.6	0.82	0.81	0.81
LingPipe + Filter	*BCN* = 10, ConfThr = 0.7	0.79	0.84	0.81

As could be seen from the table, the Analysis Engine combination *GeneLingPipeAnnotator* + *GeneMentionFilter* with parameter *BestChunkNumber* = 10 and *ConfidenceThreshold* = 0.6 achieves the highest f-measure with a good balance between recall and precision.

1 To keep the table concise, we use some abbreviations: *AE* stands for *Analysis Engine*, *StanfordNLP* stands for

Besides, although our *GeneMentionFilter* supports regular expression based filtering, we find it's really hard to find any patterns among the FalsePositive outputs. Thus,we leave the *FilteringRegularExpressions* parameter in *GeneMentionFilter* blank.

Finally, we configure the CPE with the following setting:

- Collection Reader: *LineBreakCollectionReader*
- Analysis Engines: *GeneLingPipeAnnotator* + *GeneMentionFilter*
  - *GeneLingPipeAnnotator*: set *BestChunkNumber* = 10
  - *GeneMentionFilter*: set *ConfidenceThreshold* = 0.6
- CAS Consumer: *AnnotationPrinter*[2]

2 As AnnotationEvaluator could not evaluate new inputs, it is not contained in the CPE.