Homework 1/11791

October 15, 2012 Bo Yang/byang1

1 NER Implementation

1.1 Type System

The type of annotation processed in my named entity recognizer is NamedEntity. It extends the Annotation in UIMA and has four useful attributes (start, end, id and content). The start is the number of non-whitespace characters in the processed sentence preceding the first character of the named entity, and the end is the number of non-whitespace characters in the processed sentence preceding the last character of the named entity. The id is to record the sentence identifier of the line to which the named entity belongs. The content is the string representation of the recognized named entity. Given the sentence below and the named entity "alkaline phosphatases":

P00001606T0076 Comparison with alkaline phosphatases and 5-nucleotidase.

Then, the corresponding (start, end, id, content) is:

(14, 33, "P00001606T0076", "alkaline phosphatases")

1.2 Main Components

In my named entity recognizer, there are three main components, FileSystemCollectionReader, SimpleGeneAnnotator and AnnotationPrinter.

1.2.1 FileSystemCollectionReader

FileSystemCollectionReader is modified based on UIMA example code. By giving it a InputDirectory, it first decide whether the path is a file or directory. If it is a file, FileSystemCollectionReader directly adds the file into the file list to be processed. Otherwise, FileSystemCollectionReader spans the whole directory and adds all files in the directory to the file list to be processed.

Later, FileSystemCollectionReader will be called on its getNext method. In getNext, FileSystemCollectionReader will push one file from the file list, extract the content

of the file out and put the content into Cas.

1.2.2 SimpleGeneAnnotator

SimpleGeneAnnotator is an annotator to extract GeneTag corpus from raw sentences provided in offset representation. In UIMA framework, raw data is stored in Cas. By getting raw document data from Cas, SimpleGeneAnnotator first divides the document into sentences. Then, each sentence is further divide into sentence identifier and content sentence to be annotated. In SimpleGeneAnnotator, I used the ne-en-bio-genetag model provided by lingpipe package, which will be discussed later. Each content sentence is then sent to the model. After parsing, the model returns a chunk set containing all recognized entity. Each element in this set contains the start and end position of recognized named entity substring in the raw content sentence. This is further processed to provide (start, end, id, content) information in NamedEntity.

1.2.3 AnnotationPrinter

AnnotationPrinter prints to an output file all annotations in the CAS in the alphabetical order of id supplementing the numerical order of start with the gold-standard tagging format mentioned in the writeup. Output file is set in the initialize method to the values specified in the descriptor file. Given all annotations in Cas, AnnotationPrinter first extracts all these annotations, transforms them into an inner Result Type and puts them in a collection. This collection is then sorted according to the alphabetical order of id supplementing the numerical order of start. Finally, AnnotationPrinter prints these sorted annotations into output file in the gold-standard tagging format.

1.3 Sequence Diagram

NER includes three phases. The first phase is to get description of the CPE. In this phase, SimpleRunCPE will get description of CPE from XMLParser, which describes the configuration parameters of CPE. The second phase is to create CPE according to the description. In this phase, SimpleRunCPE calls CPEProducer to create a new CPE instance. After producing the CPE, the producer will initiate the CPE with a new CPM, which manages all activities of CPE. In the last phase, SimpleRunCPE will call process function on CPE. CPE will relay this call to CPM. When CPM gets

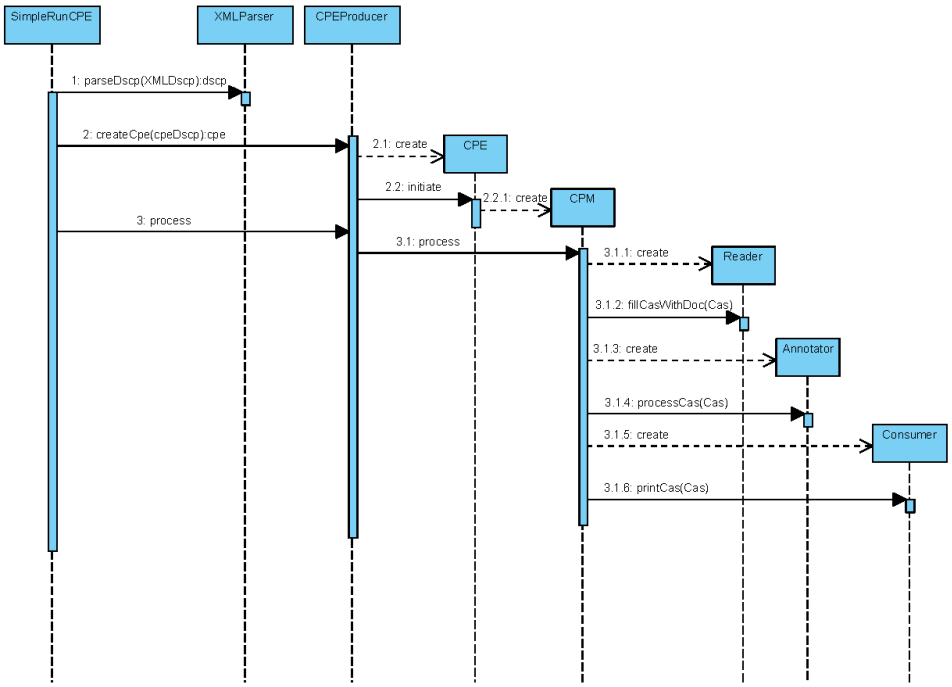

Figure 1: NER Sequence Diagram

this call, it will create CollectionReader to load files into Cas, call AnalysisEngine to annotate the Cas and let CasConsumer to print annotations to output file.

2 Machine Learning

2.1 Training

In my implementation, I used the ne-en-bio-genetag HMM Model, which is provided by the tutorial of lingpipe package. In the tutorial, it also introduced how this model was trained.

Lingpipe used GeneTag data provided by the Unites States National Center for Biotechnology Information. To train the model, tokenizer and hmmEstimator are the most important two components. For the tokenizer, lingpipe used the IndoEuropean-Tokenizer, which supports alpha-numerics, numbers, and other common constructs in Indo-European languages. With the result from the tokenizer, hmmEstimator can

estimate state emissions with sequence character language models and construct a hidden Markov model for genetag.

2.2 Evaluation

In hw1.byang1.evaluation.ResultEvaluator, I wrote a program to evaluate the accuracy of my NER. In this program, I take my output as well as the given output as arguments. Since they are both sorted by the lineId and start position of the name entity, I take one line from each output each time and compare these two strings. If these two strings match, hit count is added by 1. In the next iteration, I will take one new line from each output. If these two strings don't match, I will compare their keys. If the key of the string from my output is smaller, the wrong count is added by one and I will only take a new string from my output in the next iteration. Otherwise, the miss count is added by one and I will only take a new string from the given output in the next iteration. When there is no more new lines in either output, the loop is ended. If there are still new lines in my output, the wrong count is added by the number of remaining lines in my output. If there are still new lines in the given output, the miss count is added by the number of remaining lines in the given output. According to my statistics, my NER hit 15504 entities, missed 2761 and recognized 4670 wrong tokens as named entities.