Homework 1 Report

Lingpeng Kong
Andrew ID: lingpenk

October 16, 2012

Abstract

This document describes the design of the Named Entity Recognition (NER) system under the UIMA framework. First, we give the summary of our pipeline design and the type system we use. Then we discuss the trade-offs we made in the Collection Reader, Annotator, and Cas Consumer. We also shows the design details in these parts. Finally, performance of our NER is given.

Contents

1 Summary of the Pipeline Design

1.1 Pipeline Design

In this task, the input file we have is in raw text form, single file (for example, "sample.in"). The output we expect is also a single file which contains the original information (the sentence and the ID) plus the annotation we make on the sentence.

To achieve this goal, we designed the following simple straight-forward pipeline (See Figure 1).

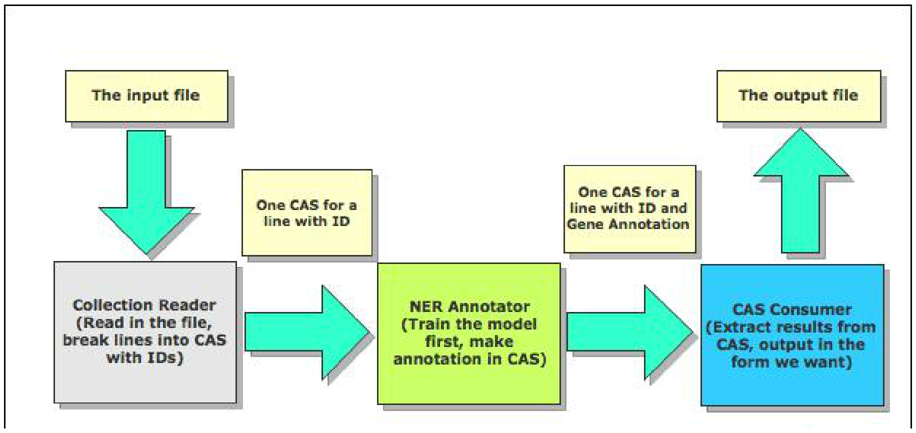
The input file
The output file
One CAS for a line with ID
One CAS for a line with ID and Gene Annotation
Collection Reader (Read in the file, break lines into CAS with IDs)
NER Annotator (Train the model first, make annotation in CAS)
CAS Consumer (Extract results from CAS, output in the form we want)

Figure 1: Pipeline Design

First, we use a collection reader to read the input file, and break it by lines. We then construct for each line a CAS, and throw it into the pipeline. The CAS will contain the ID of the sentence and the text of the sentence. Second, when we arrive the annotator, we use the LingPipe to make annotations in each of the sentences, and store them in the CAS. After this stage, we will have pretty much we want for the output. Therefore, finally, we throw these CAS objects into the CAS Consumer, where we extract the annotated strings, and the begin/end position as wanted. Then we retrieve the ID of the sentence stored in the CAS and write down them into output file.[1]

1.2 Design of Type System

The type system we design is mainly for two purposes. Since we already have the sentence string in the CAS document text, we need only to define types for the gene annotation and the ID of the sentence. The ID of the sentence should

[1]We also designed a StatConsumer to write down statistical results (precision, recall and F-Score) of the pipeline.

be associated with the CAS. Actually, it is stored only once for one sentence. That's a very important justification for design a line in a CAS rather than put all the document in one single CAS.

Therefore, in HW1TypeSystem.xml, we define model.GenTag for the gene tag, and model.SentenceInfo, for the info (ID) of the sentence. The SentenceInfo has one feature string SID to store the ID of the sentence. If the sentence has something else we need to remember in the future, this framework is easy to extend.

2 Design of the Collection Reader

In this section, we will briefly discuss the design of the collection reader. To begin with, we will first give our design of the cas, and then we will introduce how we manage to do it in the collection reader.

2.1 CAS Design

The CAS we design is to regard each line (sentence) as a CAS object. The reason we do this is 1) we do not want each CAS to be so large 2) a sentence is a perfect unit for the annotation task, since the begin/end position is for a single sentence rather than the entire input file. 3) It is easy to handle, and can be processed using multi-thread technology. 4) there should be an ID associated with each line. Using this CAS design will make this natural and easy to achieve.

2.2 Implement Details of Collection Reader

In the Collection Reader, we store the input file directory as a parameter in the XML. In initialization process, we use the parameter to construct a File object in the collection reader class.

Besides the input file, we also have a FileReader, a BufferReader, and two integers as the its variables. The FileReader and BufferReader is also initialized at the beginning. After that, we can use these two variables to read the file content line by line in the process method. The two integers here is use to store the current line we read and the total number of lines here to give information to the getProgress method to tell us where we are.

In the process method, we basically do three things. First, we read a line using the FileReader and BufferReader from the input file. Second, we break the line into two strings, namely ID and text, using the split method using space as the separator. Finally, we store the text into the CAS document text, and put the ID string in the corresponding SentenceInfo Annotation in this CAS.

3 Design of the Annotator

The logic of the annotator is pretty straight-forward. First, we initialize the LingPipe chunking we use to perform the annotation task. (The model importing should be performed only once.) Then, we get the sentence from the CAS document text and analyse if there are Gene names in it. Finally, if we find any gene names, we new a GenTag annotation, set its begin/end position, and put it into the index so that we can find it in the consumer.

3.1 LingPipe

In the annotator, we use a third-party jar "LingPipe". LingPipe is tool kit for processing text using computational linguistics. It has a pretty good NER designed for Gene name inside it.

By using the Chunk Class in LingPipe, we can easily get the beginning and ending position of a named entity. It's also pretty much we want.

The machine learning technology behind LingPipe is to using an HMM. HMM is a very common way to solving sequential problems. By using BIO tagging scheme, we transform the problem to fit the HMM's description. By mapping the beginning probability, ending probability, transition probability and emission probability into features and learning them in real data, we can then naturally apply them to solve NER problems. It is a very robust technology and has good performance in tasks like NER.

3.1.1 Dependency

To include a third-party jar in our project, we will need to do the following things:

- Add the jar into the course repository.
- Write setting.xml so that we can see the repository.
- Enable the repository in Maven.
- Add a dependency in POM.xml.

3.1.2 Resource Management in UIMA Framework

Since the LingPipe need a model file to build its Chunking object, we design in the UIMA framework a resource object for managing the medel file and provide the Chunking object the Annotator needs. (See Figure 2)

By doing this, the resource file is contained in the framework rather than leave alone outside. Actually, it is wrapped with the operation it needs to work.

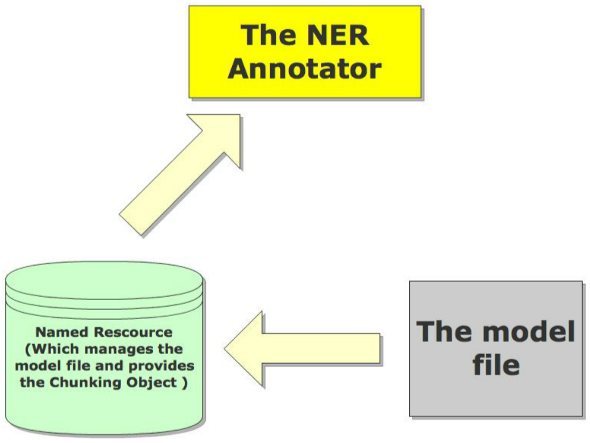
The NER Annotator
Named Rescource (Which manages the model file and provides the Chunking Object)
The model file

Figure 2: UIMA Resource Management

4 Design of CAS Consumer

4.1 The WriterConsumer

The CAS Consumer we use here is mainly to perform the task of writing the analysis results into an output file. There are something need to mention here. In the initialization, we want to find if there is a place for write. We construct the output file there based on the parameter (specify the output file's directory) in the xml file. In the process stage, things we do are very simple. We first check from the SentenceInfo what ID the sentence have. Then we check out all the gene annotations in the GenTag annotation index. For each GenTag annotation, we compute the real begin/end position (defined as the task rather than using the substring position) and write it together with the ID and words (extract from the document text of the CAS using the annotation). The writer should be closed at last.

4.2 The StatConsumer

In the StatComsumer, we first build a HashSet for the golden results. Then when each CAS (with annotation) comes, we check if that is in the golden set. By doing that, we can easily calculate the precision, recall and F-Score. (The golden set file and the output file is stored as two parameters in the Consumer.)

5 Results

The result we get using this framework is as following:

* Precision: 0.7685139288192724

* Recall: 0.848836572679989

* F-Score: 0.8066807148989308