11791-Design and Engineering of Intelligent Information System

Homework1-Report

Name: Yiwen Chen

Andrew ID: yiwenche

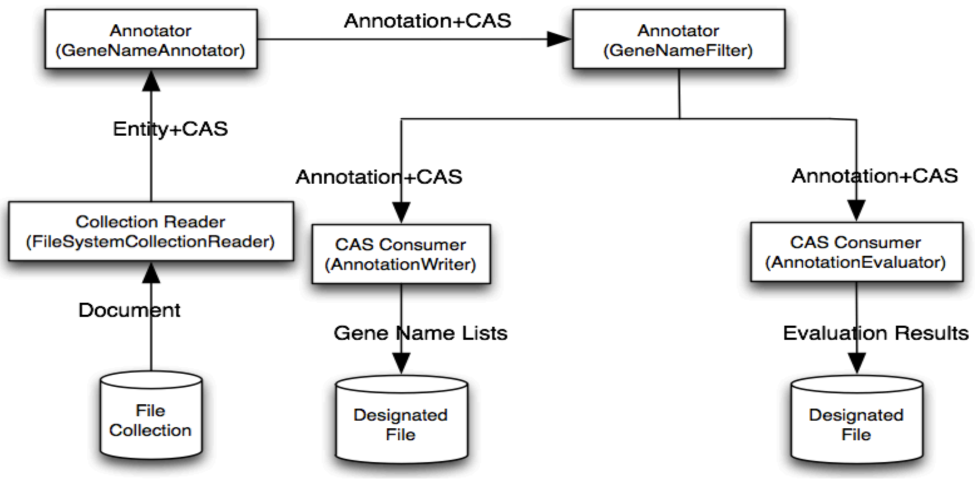
Email: yiwenche@cs.cmu.edu

1. System Overview

In this homework, I implement a Named Entity Recognition (NER) system based on UIMA (Unstructured Information Management Architecture) architecture to detect gene names from the input text file. The Collection Processing Engine (CPE) of the system implements a Collection Reader to read files from designated directory and returns corresponding CAS (Common Analysis Structure), which contains the documents to be analyzed. Analysis Engine contains two annotators, the first annotator (*GeneNameAnnotator*) calls the well-trained gene name detector tool from *Lingpipe* to detect every gene name from the input document. The annotator creates an annotation for each detected gene name and adds it to the indexes. The next annotator (*GeneNameFilter*) gets all the annotations detected by the previous annotator as input, it then use regular matching rule to discard some obvious wrong annotations from the index. There're two CAS Consumers in the CPE, namely *AnnotationWriter* and *AnnotationEvaluator*. The former reads annotations from the index and write them into target file in specific format, the other does some simple evaluations of the NER results (eg, precision, recall, f-measure).

2. System Workflow

Below is the system's workflow diagram, which describes the data flow of CPE.


Annotator (GeneNameAnnotator)
Annotation+CAS
Annotator (GeneNameFilter)
Entity+CAS
Annotation+CAS
Annotation+CAS
Collection Reader (FileSystemCollectionReader)
CAS Consumer (AnnotationWriter)
CAS Consumer (AnnotationEvaluator)
Document
Gene Name Lists
Evaluation Results
File Collection
Designated File
Designated File

3. System Design

In this chapter I will briefly introduce the design and implementations of the key components of the system in order.

3.1 Type System Design

I only define one type in this system, called *GeneName*, which inherits the super type Annotation of UIMA. Besides the Begin and End feature inherited from the super type, *GeneName* has three additional features: The first one is ID, which indexes the sentence which contains the specific gene name; The second one is Name, which records the name of the detected gene; The last feature is Accumulate, which records the total length of the previous sentences before the specific sentence that contains the gene name. That means, informally, for certain *GeneName* object g, *g.Begin* – *g.Accumulate* = the offset of the start character of *g.Name* in the sentence *g.ID*. Similarly, *g.End* – *g.Accumulate* = the offset of the end character of *g.Name* in the sentence g.ID.

3.2 Collection Reader Design

The Collection Reader I implemented is called *FileSystemCollectionReader*. It takes the parameter from user, which designates the directory the Reader will read from. If the directory contains multiple files, it will read all of them.

3.3 Analysis Engine Design

3.3.1 Gene Name Annotator

The first annotator, called *GeneNameAnnotator,* makes use of the *Lingpipe GeneTag* Model (*ne-en-bio genetag.HmmChunker*) to detect gene names in the sentences of the given document. Specific *Lingpipe* model and core package can be downloaded from the following website:
http://alias-i.com/lingpipe/web/models.html
http://alias-i.com/lingpipe/web/download.html

This model not only detects the names of genes in the text, but also finds their begin and end offsets in the text, which the annotator can use to build corresponding annotations with really simple processing. Since the sentences in the document have clear and consistent format, it's also easy to extract sentence IDs and calculate the length of sentences, which will finish the building of a specific annotations before adding it to the indexes.

3.3.2 Gene Name Filter

The second annotator contained in the Analysis Engine is *GeneNameFilter*. It uses the output of the first annotator as input, discards some annotations from the indexes if they satisfy one of two following features:

1. The name of the annotation contains only one lower case letter.
2. The name of the annotation only contains numbers.

From observation of the sample output, none of the correct annotations satisfies these requirements. *GeneNameFilter* uses regular matching rule to implement the filtering task.

3.4 CAS Consumer Design

3.4.1 Annotation Writer

The first CAS Consumer of the system is *AnnotationWriter,* which simply reads the annotations from the indexes and writes them to the designated file. User needs to enter parameter to assign the file path. Note that the output of each annotation complies with the format requirement of this homework.

3.4.2 Annotation Evaluator

The second annotator (*AnnotationEvaluator*) does some basic evaluations of the output results. It uses the file *sample.out* as benchmark, reading all the annotations from it into a hash table. Then the evaluator begins to read annotations from the indexes, each time it checks if the annotation detected by the annotator is a correct one which has already been stored in the hash table. Finally, it calculates the precision, recall and f-measure of the system and writes these metrics to the designated file, whose path should also be entered by the user.

4 System Performance

Evaluation results of the system are listed below:

Total Number of Gene Name Annotations in Standard File: 18265
Total Number of Gene Name Annotations CPE find: 20107
Total Number of Gene Name Annotations which are found to be correct: 15504

Precision: 0.7710747500870344

Recall: 0.848836572679989
F-measure: 0.8080892317314708

5. System Strengths and Weakness

The system is based on UIMA architecture, which has a well-defined structure and clear workflow. In addition, it also has good error handling mechanism. However, there are still some weaknesses that can be improved in the future:

1. The system only uses one NER method. It's better to try different algorithms to improve the final results.
2. In the evaluation process, the *AnnotationEvaluator* reads all the sample annotations from the disk into memory, in order to improve the evaluation speed. However, if the sample file becomes much larger, it might be impossible to store it in memory as a whole. Other methods should be applied in that situation.
3. In this system, the two CAS Consumers (*AnnotationWriter, AnnotationEvaluator*) run sequentially in the workflow. However, they don't have any dependency relationship. So it's better to make them run in parallel when the running time of each consumer is very long.