HW 1 REPORT
KARISHMA KANAL
ANDREW ID: kkanal

1 Introduction

The GeneTagging System is an information extraction system which deals with named entity extraction of gene and gene product mentions in a given text. It makes use of the Unstructured Information Management Architecture (UIMA) capabilities. The following document provides an insight into the system with respect to various architectural and algorithmic aspects.

2 Basic Input Output formats used in the System

2.1 Input

On a higher level the system takes as input a file which consists of ascii sentences, one per line. The format of sentences in the input file is as follows:

Sentence-Id text

For eg: P00001606T0076 Comparison with alkaline phosphatases and 5-nucleotidase

2.2 Output

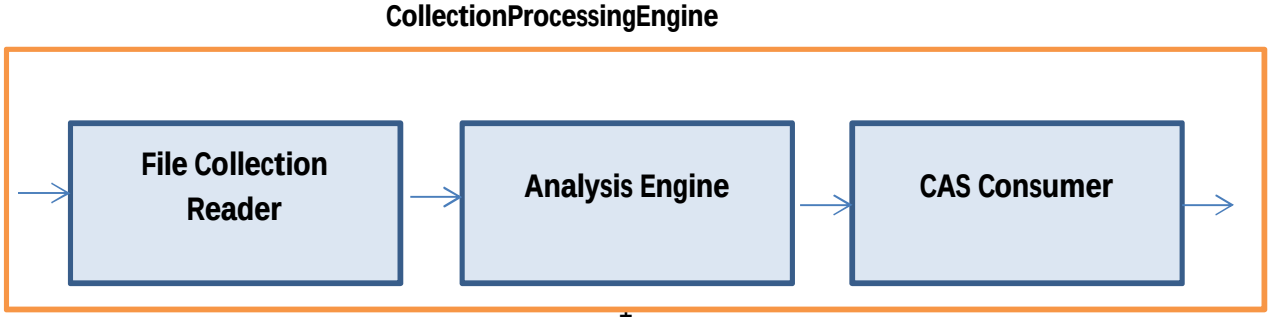
Output is a file consisting of an ascii list of reported gene mentions, one per line and formatted as follows:

sentence-identifier-1|start-offset-1 end-offset-1|optional text...
If a sentence in the input file has two or more gene mentions, each is outputted as a separate line as seen in the example below:

P00001606T0076|14 33|alkaline phosphatases
P00001606T0076|37 50|5-nucleotidase

The start-offset is the number of non-whitespace characters in the sentence preceding the first character of the mention, and the end-offset is the number of non-whitespace characters in the sentence preceding the last character of the mention.

3 Basic Components of System



3.1 Type System

The Type System is a user defined schema that identifies all the data that will be produced by the annotator and between the various components.

In our system the Type System defines two CAS annotations:

3.1.1SentenceAnnotation
This annotation is used to store the contents of the file line by line. It consists of a single feature called Sentence, which holds the line read from the input file.

3.1.2 GeneAnnotation
This annotation stores details of gene mentions found in the text. It has several features:

3.1.2.1 Sentence_ID: The sentence Identifier got from the input sentence

3.1.2.2 Start_Index: Start off-set position of found gene mention in text

3.1.2.3 End_Index: End off-set position of found gene mention in text

3.1.2.4 GeneName: The gene mention found in the text.

3.2 File Collection Reader

The File Collection Reader is responsible for reading the file and outputting each line of the file as a SentenceAnnotation. It consists of the Java Class file for the reader and xml descriptor for the same.

3.3 Analysis Engine

This component analyzes the sentence provided and then ouputs a GeneAnnotation containing the required information of the gene mention found.

3.4 CAS Consumer

The CAS Consumer takes as input(consumes) the enriched CAS provided by the Analysis Engine and produces the output file in desired format.

3.5 Collection Processing Engine

The functionality defined in the Collection Processing Architecture is implemented by a *Collection Processing Engine* (CPE). A CPE includes an Analysis Engine and adds a *Collection Reader*, a *CAS Initializer* (deprecated as of version 2), and *CAS Consumers*. The part of the UIMA Framework that supports the execution of CPEs is called the Collection Processing Manager, or CPM.[1]

4 Tools used for analysis

At the heart of the system is the Analysis Engine which analyzes the text to find gene mentions. To achieve this, a software tool called ABNER has been used. ABNER is a software tool for molecular biology text analysis. At ABNER's core is a statistical machine learning system using linear-chain conditional random fields (CRFs) with a variety of orthographic and contextual features.[2]

The JAVA API for ABNER has been used to implement the logic for analysis in our AE. The Tagger object, trained on BioCreative corpus, of ABNER is used. It tags all entities of the type "PROTEIN".

No other training data, machine learning or NLP tools have been used in the system.

5 Data Flow in the System

As described in Section 2 and Section 3, the system takes as input a file. This file is read line by line by the FileCollectionReader. This is acheieved by basic Java file reading operations. It outputs SentenceAnnotations, containing the read lines. These Annotations act as input for the Analysis Engine, which uses the ABNER tool to identify gene mentions and ouputs the GeneAnnotations. The CasConsumer takes the GeneAnnotations and writes the contents to the output file using basic java file writing operations.

6 Design Patterns in System

6.1 Creator Pattern

According to the creator pattern a class has a responsibility of creating an object if any one of the following conditions is true:

B "contains" or compositely aggregates A.

B records A.

B closely uses A.

B has the initializing data for A.[3]

In this system, this pattern is used to assign to the FileCollectionReader the responsibility of creating the SentenceAnnotation as it has the initialization data for the object. Similarly AnalysisEngine has the responsibility of creating the GeneAnnotation as it has the initialization data for the object.

6.2 Information Expert Pattern

According to the information expert pattern you assign a responsibility to the class that has the information needed to fulfill it.[4]

In this system, each class is assigned specific responsibilities using this pattern. The FileCollectionReader has the responsibility to read the file as it has access to the file. Similarly the AE contains the NER for analysis and hence was assigned the same.

6.3 Low Coupling

You need to assign responsibilities to classes so that (unnecessary) coupling remains low. This was implemented in the system by keeping each responsibility specific as explained in Section 6.2.

6.4 Controller Pattern

According to the controller pattern, assign the responsibility of receiving and coordinating beyond the UI layer to an object representing one of these choices:
Represents the overall "system," a "root object," a device that the software is running within, or a major subsystem (these are all variations of a facade controller).
Represents a use case scenario within which the system operation occurs (a use case or session controller)[5]

This pattern has been implement in the UIMA framework which provides the Collection Processing Manager which supports the execution of the Collection Processing Engines.

6.5 Polymorphism

According to polymorphism when related behaviors vary by type you assign responsibility for the behavior to types using polymorphism.[6]

This pattern can be seen at various places in the system. One example would be the Iterator class which handles various types.

7 Alternate tool used

One alternate method used to identify the gene mention that was used was to use the Stanford NLP provided and then pass the results obtained doing so to ABNER to identify gene names from them and discard other entries. However this method did not provide for spans in the format needed i.e it considered blank spaces too. Also the number of gene mentions found was half as compared to those got by ABNER. Thus performance without the Stanford NLP was better and hence only ABNER was used.

8 Limitations and Improvements

The system is limited in its capability to find gene mentions. An improvement to the system could be to use a better NER along with training more data.

NOTES

1. Source: UIMA Tutorial and Developers Guide
   http://uima.apache.org/downloads/releaseDocs/2.1.0-incubating/docs/html/tutorials_and_users_guides/tutorials_and_users_guides.html#ugr.tug.aae.getting_started
2. Source: ABNER Website
   http://pages.cs.wisc.edu/~bsettles/abner/
3. Source: Applying UML and Patterns – Larman
4. Source: Applying UML and Patterns – Larman
5. Source: Applying UML and Patterns – Larman
6. Source: Applying UML and Patterns – Larman