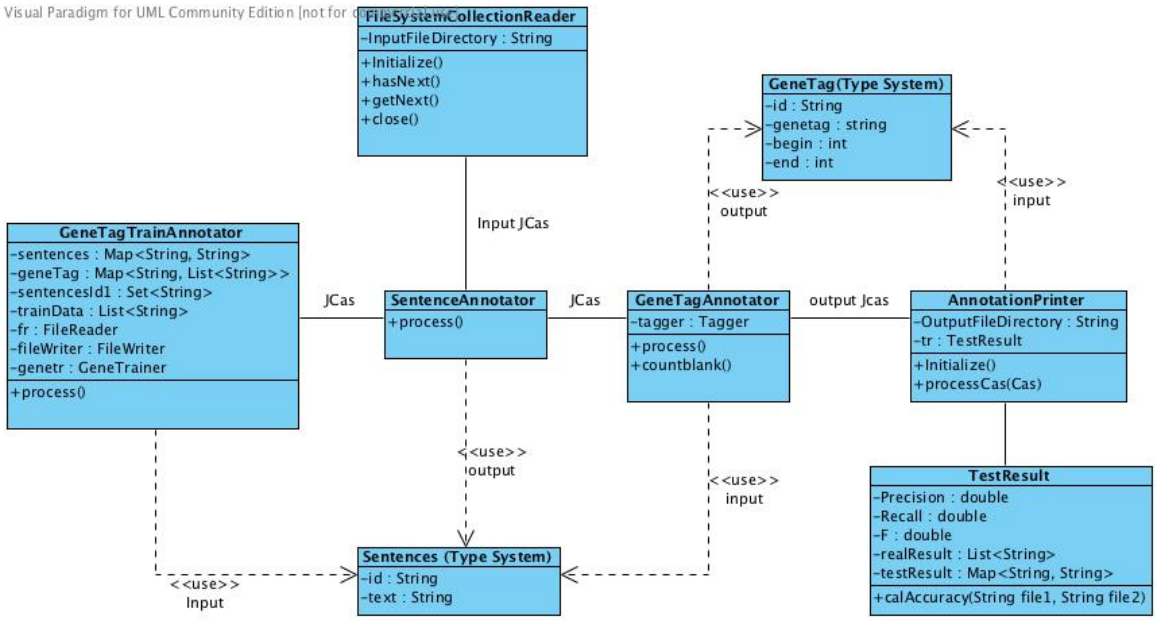
11-791 HW1-Report Pengqi Liu

System Design

The *Unstructured Information Management Architecture* (UIMA) is an architecture and software framework for creating, discovering, composing and deploying a broad range of multi-modal analysis capabilities and integrating them with search technologies. Thus in this project, I'll use UIMA to build a Gene Mention Tagging system which can recognize Gene related entities in the text.

In UIMA, the *Collection Processing Architecture* defines components for reading raw data formats from data collections, preparing the data for processing by Analysis Engines, executing the analysis, extracting analysis results, and deploying the overall flow in a variety of local and distributed configurations. From this definition, I designed three main components in my CPE system: Collection Reader (read from file), Analysis Engine (process data) and Cas Consumer (output the result). I will discuss each of them in detail in the following sections. The UML of my class structure is shown below:



Collection Reader

The name of my collection reader is FileSystemCollectionReader, which can use basic java I/O to read text from a given file directory and put it into JCas for analysis engines to use.

Analysis Engine

Type System

I defined two type systems in my project:

- Sentences: extends Annotation (with begin and end), plus another two new features
    - id: id of the sentences
    - text: text of the sentences
- GeneTag: extends Annotation (with begin and end), plus another two new features
    - id: sentence id of the gene tag
    - geneTag: gene tag we want to print out

These two type systems are used as my annotators' input and output, which will be explained later.

Annotator

In my project, I defined three annotators to complete this task:

- SentenceAnnotator:
    - Function: split whole text into single sentences and save in **Sentences Type System**. JCas => Sentences => JCas
    - Output: **Sentences**
- GeneTagAnnotator:
    - Function: Read sentence from **Sentences**, use the training model to find the gene tag with their location and save in **GeneTag Type System**. JCas => Sentences => GeneTag => JCas
    - Input: **Sentences**
    - Output: **GenTag**
- GeneTagTrainAnnotator
    - Function: use the sentence in **Sentences** and gene tag in sample.out file to construct a specific document format for training samples. Then I use external functions to train these data and output a training model. JCas => Sentences => Training model
    - Input: **Sentences**, sample.out
    - Output: geneModel1.crf.gz

Cas Consumer

The name of my Cas Consumer is called AnnotationPrint, which can print out **GeneTag** into specific output files (hw1-pengqil.out). Besides, I run a testResult program in this Cas Consumer to calculate the precision, recall and F value of my tagging document.

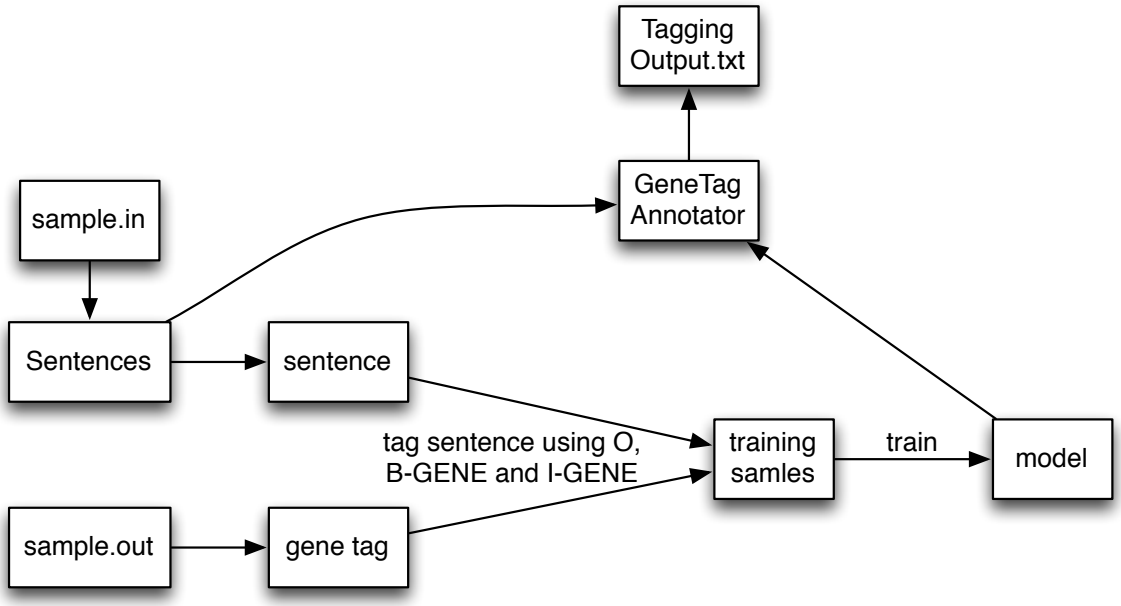
Algorithm Design

Main Idea

In my project, I used a machine-learning technique called BioTagger to tag gene words. The main idea is using the probabilistic sequence-tagging framework of **conditional random fields**, which is raised by *Ryan McDonald* and *Fernando Pereira* from UPenn.

They tag the words in a text using three different taggers: O, B-GENE, and I-GENE. O means that this word is not a gene word, B-GENE means that this word is the first one of a gene words and if there are following gene word, we use I-GENE to tag them. For example, in "Comparison with alkaline phosphatases and 5-nucleotidase", "alkaline phosphatases" and "5-nucleotidase" is gene words, so we tag this sentence as "Comparison O, with O, alkaline B-GENE, phosphatases I-GENE, and O, 5-nucleotidase, B-GENE". After tagging the training sample like this, they use conditional random fields to train the data and output a training model for later use.

Implementation



The data flow can be summarized as above. I first use sample.in (Sentences Type System) and sample.out to tag the sentence using the method I mentioned before and got training samples (saved as src/ main/ resources/ data/ training/ data/ training_data.txt). Then I use this training samples to run external training method to generate my training model (saved as src/ main/ resources/ mode/ geneModel1.crf.gz). Since I have already trained my model, thus there is no need to train it again since it's very time costly (about an hour). At last, in my GeneTagAnnotator, I can use this generated training model to tag my sentences.

External library

I used 10 external jar libraries for my training method. bioTagger is the one that implement the core algorithm of training and tagging. Other jas like mallet-no_gnu, opennlp-tools-1.0.0, maxent, trove, mallet-deps, ant-nodeps, ant, crabapple-1.0, wordfreak-2.2 are supporting my algorithm. I first met some problem finding and building these jars to Maven Dependency though I have uploaded to the server. Finally I found out that I can achieve it by directly changing the pom.xml file.

Preliminary experiments testing

I use the sample data to test my training model and have the following result:

- Precision:0.7019211983067405
- Recall:0.5900903367095538
- F:0.641165972635336

The result does not seem to be very good. I guess the reason might be that the process of generating training data from sample.in and sample.out has some mistakes. This can be improved in the future assignments.