Hw2 Logical Architecture and UIMA Analysis Engines Design & Implementation

1 Overview

Basically, this report introduces the system design, especially the three annotating approaches and the merging voting process. I use an aggregate analysis engine to combine all the annotators.
The following sections explain the details about this system, like system architecture, core techniques and algorithms, and performance evaluation based on the sample output.

2 System Design

The flow chart below is an overview of system design. And the following sections will introduce each component in detail.

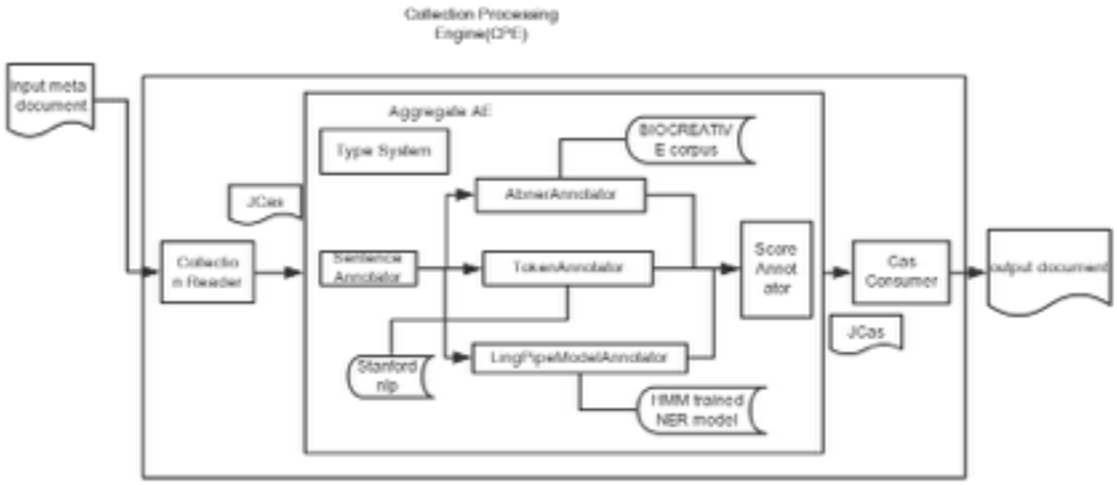


Figure 1 System Flow Chart

2.1 Collection Reader

Similar to last homework, the Collection Reader handles the input, initialize the CAS. It reads the original document and converts into a format required by the Analysis Engine. In my system, the collection reader reads the file hw2.in and turn it into string *cas* and update the document text by using jcas.setDocumentText(cas). My collection Reader could only read one file and there is a flag called *readflag=1.* After reading one document, it turns to be readflag=0 and the getNext() function returns false when *readflag=0*.

2.2 Type System

The core type in this system is the Gene which inherit from the system annotation. The *casProcessId* is to record where this *jcas* derives, which is useful for the annotator merging process. The *confidence* is to say how much percent the tagging could be correct, and is also used in the ScoreAnnotator for integration. Properties in type Gene are *id*, gene name, and the start and end position, important to casconsumer. Token, AGene, LGene, and FinalGene are to store intermediate data produced by each annotating approach. Another type is the Sentence, and it splits each sentence in the original document and provides id for gene types.

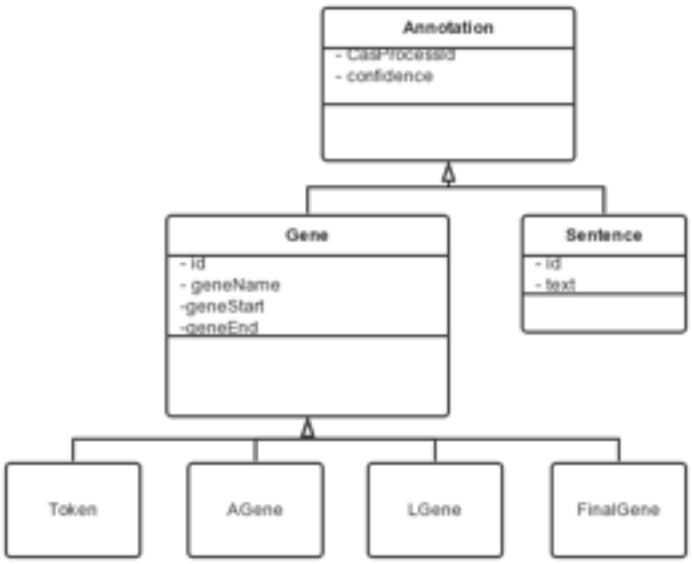


Figure 2 Class Diagram of Type System

2.3 Annotator

2.3.1 SentenceAnnotator

First, it populates the CAS generated by Collection Reader in the previous step. It also does the job of splitting up each line into two parts by using the first white space, one is for the sentenceId i.e. *P00001606T0076*, and the other is the sentence text like *Comparison with alkaline phosphatases and 5-nucleotidase*. After that, it again populates the CAS, to update the information of type Sentence in Jcas, that is to add the indexes for each sentence.

2.3.2 TokenAnnotator

In this annotator, I combine both **stanford nlp** libraries and a **gene name database** crawling from the website(http://www.genenames.org/cgi-bin/download). The PosTagNamedEntityRecognizer recognized nouns which are retrieved in the database to see if it is a gene name. As for the database, I choose approved gene name, approved and previous gene symbol, and synonyms. After that, I create file geneDB.tag, and clean it offline into a specific format file, geneDBcleaned.tag. The system reads this file and store the gene names into HashSet for retrieval. The results are stored in the token type. Because each name generated by this annotator has been found in the gene database. They could definitely be gene names, although some names are not used nowadays. Therefore, I assumed that its confidence is 1.0.

2.3.3 LingPipeModelAnnotator

Different from last homework, this time, I implement the **Confidence Named Entity Chunking** approach, since it provides confidence for each recognized gene name. It's interesting to find out how the results of nBest method derive by observing each candidate's confidence.
In order to get a better result, I introduce some heuristics rules to clear some noises like, unmatched brackets, numbers. More details are in the *clearHeuristic* method.
Results show that names with confidence more than 0.99 could be considered as gene, and if it is between 0.5 and 0.99, it has a high chance to be genes.
Here are some testing results of continuously modify the confidence condition and we could see the tradeoff between recall and precision rate.
Actually, I have tested a lot about this low boundary. If it is higher than 0.6, it could also increase a little. However, I thought it will be overfit if I increase this boundary. Another reason for choosing 0.5 is to better integrate these three annotators and compare their impact for the final results. LingPipe and Abner are both statistical model and they could probably be overfit with small training data. What's more, Abner doesn't give us confidence of their result. So, I assign the same confidence for each word recognized by Abner as 0.5, the same as this Lingpipe low boundary.
confidence low boundary: 0.3

```
// ########correct##########15998
// #####recall#########0.8758828360251848
// #####precision#########0.7452715922854747
// #####f-score#########0.8053157484080441
// #####geneName count#########21466
```

confidence low boundary: 0.5

```
// ########correct##########15265
// #####recall#########0.8357514371749247
// #####precision#########0.8057960304054054
// #####f-score#########0.8205004165658846
// #####geneName count#########18944
```

nbest last time:

```
// ########correct##########15504
// #####recall#########0.848836572679989
// #####precision#########0.7685139288192724
// #####f-score#########0.80668068
// #####geneName count#########20174
```

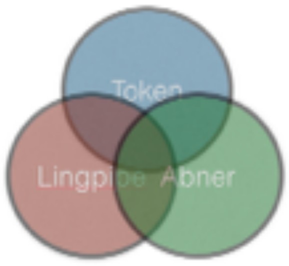
2.3.4 AbnerAnnotator

Here, I used the *BIOCREATIVE corups* given in Abner, and assigned the confidence of each result as 0.5 due to the reason mentioned above. Here is how it works.

```
Tagger t = new Tagger(Tagger.BIOCREATIVE);
String[][] result = t.getEntities(text);
for (int i = 0; i < result[0].length; i++) {
String content=result[0][i];
}
```

2.3.5 ScoreAnnotator

The heart of this Annotator is an integration and voting process. Confidence values of each method are used as voting weights. For example, if the name is generated by TokenAnnotator, we give a score of 1.0. Likewise, name from AbnerAnnotator has an average score of 0.5. We combine these score by summing up if they show up in more than two annotators. Here is a diagram showing their relationship.



FinalGene type is used to store the gene derived from these three methods. However, if gene name is only generated by one annotator, its *casProcessId* will be assigned as its annotator's name like *TokenAnnotator*. And, if the name is from more than two annotators, its *casProcessId* will be *ScoreAnnotator*. This initiative could help evaluate the possibility of being a gene name. What's more, according to LingPipe's high performance when its confidence is bigger than 0.99, I also include them into the final results.

These analysis engine above compose the aggregate AE, and they follow fixed flow sequence, populates the CAS.

2.4 Cas Consumer

Finally, CAS consumer generates the results of CPE in a specific format which is set in its java file. It retrieves all the FinalGene populated by AE, and writes to a pre-configured file. To, improve the results, we just use the finalGene whose *casProcessId* shows it is from the integration annotator ScoreAnnotator as the final results. Although, adding the result from lingpipe will give a better performance for this sample.in, I still use the combined result to avoid overfit. And, this interesting issue will be discussed in the following sections.

3 Techniques, Algorithms, and Design Patterns

3.1 Tricks of Clearing Dataset

The gene name dataset given in my system is crawled from website, and its original format is not good enough and has many duplicate values. So, by using some tricks to deal with the strings, we could see the good result in geneDBcleaned.tag. The original dataset and the intermediate data are separately stored in approveData and geneDB.tag in the dataset repository.

3.2 Lingpipe's Confidence Named Entity Chunking

This is the only approach I found out has confidence for each tagging result. By learning this approach, I got the idea of how the First-Best Named Entity Chunking and N-Best Named Entity Chunking work. Here is a brief overview of confidence named entity chunking.

```
ConfidenceChunker chunker = null;
    try {
      chunker = (ConfidenceChunker) AbstractExternalizable.readObject(modelFile);
    } catch (ClassNotFoundException e) {
      // TODO Auto-generated catch block
      e.printStackTrace();
    } catch (IOException e) {
      // TODO Auto-generated catch block
      e.printStackTrace();
    }
Iterator<Chunk> chunkit = chunker.nBestChunks(cs, 0, cs.length, MAX_N_BEST_CHUNK);
      for (int n = 0; chunkit.hasNext(); n++) {
        Chunk c = chunkit.next();
        double conf = Math.pow(2.0, c.score());
        int start = c.start();
        int end = c.end();
        String phrase = text.substring(start, end);
     }
```

By the way, instead of using the given gene corups, I found another corups with more gene names, and then trained a new model (train-with-more-gene.HmmChunker) using LingPipe's method. Here is how I trained the new model.

```
static final int MAX_N_GRAM = 8;
static final int NUM_CHARS = 256;
static final double LM_INTERPOLATION = MAX_N_GRAM; // default behavior

public static void main(String[] args) throws IOException {
    File corpusFile = new File(args[0]);
    File modelFile = new File(args[1]);

    System.out.println("Setting up Chunker Estimator");
    TokenizerFactory factory
      = IndoEuropeanTokenizerFactory.INSTANCE;
    HmmCharLmEstimator hmmEstimator
```

```
      = new HmmCharLmEstimator(MAX_N_GRAM,NUM_CHARS,LM_INTERPOLATION);
    CharLmHmmChunker chunkerEstimator
      = new CharLmHmmChunker(factory,hmmEstimator);

    System.out.println("Setting up Data Parser");
    GeneTagParser parser = new GeneTagParser();
    parser.setHandler(chunkerEstimator);

    System.out.println("Training with Data from File=" + corpusFile);
    parser.parse(corpusFile);

    System.out.println("Compiling and Writing Model to File=" + modelFile);
    AbstractExternalizable.compileTo(chunkerEstimator,modelFile);
}
```

3.3 Heuristic Rules to Clear Noises

After observing the results of lingpipe and their confidence, I found out some interesting rules. If the confidence is more than 0.99, it has the biggest chance to be a gene name. As for the names with confidence between 0.5 and 0.99. I found out some exceptions are numbers, one letter word, and some has unmatched brackets (A paragraph about gene usually use a lot of parentheses for detail explanations). They are definitely not gene names, so I use some conditions to rule them out.

The following generally do not qualify as gene name mentions:

1. Generic terms

zinc finger alone (but zinc finger protein is an accepted gene/protein mention)

2. Mutations

p53 mutations

3. Parens at start and end which ‘wraps’ the whole gene name

(IGG)

4. TATA boxes
5. Receptors: if a receptor is specified, the gene name without “receptor” is not considered to be a valid alternative.
6. Synonyms: if a synonym is given in the sentence which implies certain words are necessary to the gene name, they will be required in the alternatives

For rabies immunoglobin (RIG), “immunoglobin” alone will not be a valid al- ternative because RIG implies that “rabies” is part of the name in this context.

7. Non-peptide hormones
8. DNA and protein sequences

3.4 Voting Process

This is the heart of the whole system and the most critical thing is to choose the weight values of each method.

One thing that confused me all the time is I could not find any confidence value in Abner approach. And comparison of its results and lingpipe’s clarifies they are quite similar. I assign the same confidence as the low boundary of lingpipe. In this case, if the final sum score is more than 1.0, it must come from at least two Annotators. This is a threshold to reject low-quality annotating result generating by only one annotator method.

3.5 Design Pattern

As for the design pattern, in this system, each class is assigned specific responsibilities using Information Expert Pattern. For example, CollectionReader is responsible for reading the input file. Another pattern is the creator pattern. Again, the CollectionReader initializes the CAS for class SentenceAnnotator. It' worth to mention that my system aims to reduce coupling as well as to increase cohesion.

4 Evaluation and Improvement

4.1Alternative Solutions and Interesting Findings

Since last time, I tried to both first use PosTagNamedEntityRecognizer and then the lingpipe and use them in a reversed order. Results showed that the process of firstly recognizing nouns would rule out many compound gene names, and if using after lingpipe method, it will split up the perfect gene names into parts. That's why it has a bad performance on recognizing the gene names. Based on this, I tried to use these methods concurrently, and merged the result in the voting process.
I have tried several third-party named entity recognition involves the supervised training of a statistical model like LingPipe, StanfordNER, ABNER. And you could find my trained model in the dataset repository, such as ne-en-bio-genetag.HmmChunker, ner-model.ser.gz, ab-test-model.ser.
As for LingPipe, the model result is the same as that given on its website, because we just use the same training dataset. Here are some codes for training model. The corpus file is required to have this format, like “P00010943A0733
Flurazepam_TAG thus_TAG appears_TAG to_TAG be_TAG an_TAG effective_TAG hypnotic_TAG drug_TAG with_TAG the_TAG optimum_TAG dose_TAG for_TAG use_TAG in_TAG general_TAG practice_TAG being_TAG 15_TAG mg_TAG at_TAG night_TAG ._TAG”.

```
static final int MAX_N_GRAM = 8;
static final int NUM_CHARS = 256;
static final double LM_INTERPOLATION = MAX_N_GRAM; // default behavior

public static void main(String[] args) throws IOException {
    File corpusFile = new File(args[0]);
    File modelFile = new File(args[1]);

    System.out.println("Setting up Chunker Estimator");
    TokenizerFactory factory
      = IndoEuropeanTokenizerFactory.INSTANCE;
    HmmCharLmEstimator hmmEstimator
      = new HmmCharLmEstimator(MAX_N_GRAM,NUM_CHARS,LM_INTERPOLATION);
    CharLmHmmChunker chunkerEstimator
      = new CharLmHmmChunker(factory,hmmEstimator);
    System.out.println("Setting up Data Parser");
    GeneTagParser parser = new GeneTagParser();
    parser.setHandler(chunkerEstimator);

    System.out.println("Training with Data from File=" + corpusFile);
    parser.parse(corpusFile);

    System.out.println("Compiling and Writing Model to File=" + modelFile);
    AbstractExternalizable.compileTo(chunkerEstimator,modelFile);
}
```

Second, similar to lingpipe, stanfordNER packaged in stanford-nlp 3.4 also ask for specific data format as corpus. Here is an example.

```
Studies		TAG
on	TAG
immunoglobulin		GENE2
E	GENE2
:	TAG
the	TAG
impact	TAG
.	TAG
```

However, I did not use this since it has a bad performance on testing dataset. It shows result, like this.

<GENE2>Studies</GENE2> <TAG>on</TAG> <GENE1>immunoglobulin E</GENE1><TAG>: the</TAG> <GENE2>impact of a sojourn</GENE2> <TAG>with</TAG> <GENE2>Professor Dan H.</GENE2>
<TAG>an</TAG> <GENE2>Peroxydase reaction stains were negative</GENE2><TAG>,</TAG> <GENE2>chloroacetate esterase</GENE2> <TAG>were strongly positive.</TAG>

Actually, better results(showed below) trained by the same model appear if using stanford-ner.jar, stanford-ner-2013-11-12.jar, stanford-ner-3.3.0-javadoc.jar, and stanford-ner-3.3.0-sources.jar instead of stanford-corenlp.jar on maven repositories. They could be downloaded from the website http://nlp.stanford.edu/software/CRF-NER.shtml. Stanford NER is also known as CRFClassifier. The software provides a general implementation of (arbitrary order) linear chain Conditional Random Field (CRF) sequence models.

<TAG>Studies on</TAG> <GENE1>immunoglobulin E</GENE1><TAG>: the impact of a sojourn with Professor Dan H.</TAG>
<TAG>an</TAG> <GENE1>Peroxydase</GENE1> <TAG>reaction stains were negative,</TAG> <GENE2>chloroacetate esterase</GENE2> <TAG>were strongly positive.</TAG>

This is a point that I could improve in future work.

Finally, abner could also train model file, however I just use the given BIOCREATIVE corups.

After training my own model by various method, I found out there is a neck bottle that I have limited dataset for training. So, for improvement, I should dig out a bigger dataset. Another finding is that all these three methods to train require the training set has a paragraph with both the target gene name and its content. If we only have tagged gene names, we could not have a good result when testing on a randomly selected text.

4.2 Tradeoff for precision and recall

I have done several experiment for setting up a threshold for the voting process. All the evaluation scores are based on *hw2.in* and *sample.out* in terms of Precision, Recall, and f-score are given below.

First, if we only consider the name as gene when it is annotated by at least two annotators, we could have this precision and recall rate.

```
########correct##########9789
#####recall#########0.5359430604982206
#####precision#########0.9047971161844902
#####f-score#########0.6731536239856966
#####geneName count#########10819
```

Second, if we also add those results from LingPipe, we could have this.

```
########correct##########14973
#####recall#########0.8197645770599508
#####precision#########0.8256865556413368
#####f-score#########0.8227149097502678
#####geneName count#########18134
```

Third, if we instead add those from ABNER, here it is.

```
########correct##########10627
#####recall#########0.5818231590473584
#####precision#########0.7702399072262086
#####f-score#########0.6629031251949349
#####geneName count#########13797
```

Finally, if we instead add the result generated by the gene database, we have this.

```
########correct##########9692
#####recall#########0.5306323569668765
#####precision#########0.7373706634205721
#####f-score#########0.6171479512241713
#####geneName count#########13144
```

The final result testing by the grading script is:

```
Precision: 0.911550771021
```

```
Recall: 0.686120996441
F1 Score: 0.782931933902
```

Comparing these evaluation scores above, some conclusion arrives.

1.LingPipe gives the highest f-score on sample.in, however, its training set is just sample.in. So, it might be overfit.
2.Retrieval of gene database reduces greatly the precision. Observation from the sample.out shows that al lot of tagging names are not gene but protein. That's why this method doesn't have a good performance.
3.ABNER is quite stable on different testing set. It recognizes a lot of protein if you track it source code, you will find out that ABNER could recognize DNA and protein type. I have saved the intermediate result of ABNER in abner.out and it shows that most of the names recognized are from type protein. That's why it could increase the performance on hw2.in.
4.Based on analysis above, I finally choose the first method. It has a high precision rate, but fails to find enough gene names. However, it would not be overfit on this hw2.in.

4.3 Improvement

To improve the system, one way is to find out some other dataset for training. Another approach could be to train the sample.out by cross validation, and to use half of it as training set and the other half as testing set.

Another idea about the voting process is to use regression analysis to consider half of the sample.out as training set and the other half as a test set. With gradient descent method, we could obtain the weight of each method mentioned in this system. I tried the linear regression (details are in util\LinearRegressionTest.java), and found out the cost keeps increasing, so we could not have a good result of the coefficient of the three method, and that is to say it is hard to find a linear relationship between the three method. For its reason, I guess, the nodes have a scattered distribution. So, in this case, we might have to sort the training dataset according some rules for better linear regression. Otherwise, we could try some methods like non-linear regression. Here is some sample code about linear regression.

Repeat the following code until we get the minimum cost:

$$\boxed{\theta_j := \theta_j - \alpha \frac{\partial}{\partial \theta_j} J(\theta_0, \theta_1) \qquad (\text{for } j = 0 \text{ and } j = 1)}$$

$$\begin{aligned}
\text{temp0} &:= \theta_0 - \alpha \frac{\partial}{\partial \theta_0} J(\theta_0, \theta_1) \\
\text{temp1} &:= \theta_1 - \alpha \frac{\partial}{\partial \theta_1} J(\theta_0, \theta_1) \\
\theta_0 &:= \text{temp0} \\
\theta_1 &:= \text{temp1}
\end{aligned}$$

5 Reference

1.LingPipe tutorial
http://alias-i.com/lingpipe/demos/tutorial/ne/read-me.html
2. LingPipe demo
http://alias-i.com/lingpipe/web/demo-ne.html
3.uima example
4.deiis example given by piazza
5.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.tu g.aae.getting_started
6. About how to train model using Stanford-NER
http://blog.csdn.net/limisky/article/details/17025861
7. Gene Name database
http://www.genenames.org/cgi-bin/download
8. Stanford-NER API
http://nlp.stanford.edu/software/crf-faq.shtml#extend
9. Abner API
http://pages.cs.wisc.edu/~bsettles/abner/javadoc/
10. Machine Learning by Andrew Ng on Coursera
https://www.coursera.org/course/ml?from_restricted_preview=1&course_id=972303&r=https
%3A%2F%2Fclass.coursera.org%2Fml-006