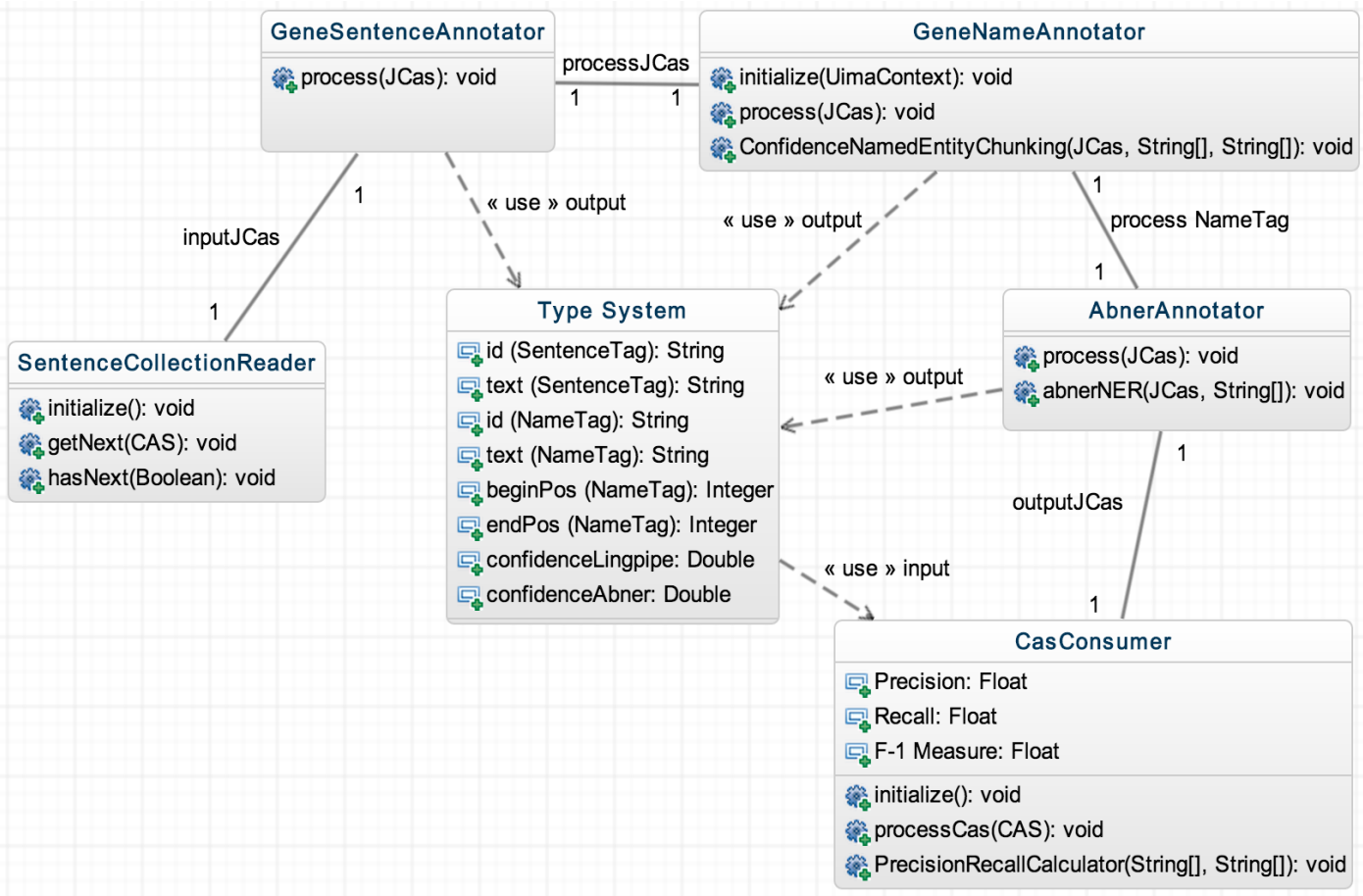
HW2-Report

1. System Architecture

I use an integrated system architecture design pattern to implement this Gene Mention Name Entity Recognizer. There are three main components in my CPE system: Collection Reader that will read file, Analysis Engine that process the input, Cas Consumer that output the result. The architecture and general data flow can be seen below:

UML Diagram:



1.1 Collection Reader (SentenceCollectionReader)

The collection reader use basic Java I/O to read the file. By using the getNext() function, it reads the input file as a string and put it into CAS for Analysis Engine usage.

1.2 Analysis Engine (GeneAE)

1.2.1 Type System

SentenceTag

id – The unique identifier at the beginning of each line;

text - The original text information for corresponding id.

NameTag

id: The mark of relation between words and sentences;

text: The gene names for further usage;

beginPos: The begin position in the sentence for each word;

endPos: The end position in the sentence for each word;

confidenceLingpipe: The confidence from Lingpipe for this word;

confidenceAbner: The parameter showing the confidence from Abner.

1.2.2 Sentence Annotator (GeneSentenceAnnotator)

There are three annotators in my implementation. Firstly here comes GeneSentenceAnnotator. It reads the string from JCas passed by collection reader, separates it into ids and text of sentences line by line, and sets feature ids and text for the SentenceTag annotation.

1.2.3 Word Annotator (GeneNameAnnotator)

The second annotator - GeneNameAnnotator implements confidence named entity chunking from LingPipe to run a statistical named entity recognizer. It imports the "ne-en-bio-genetag.HmmChunker" model file base on a dataset trained by a Hidden Markov Model. This annotator labels words with ids, text, begin positions, end positions and confidence (between 0 and 1) from LingPipe.

What's more, by utilizing regular expression, this annotator excludes meaningless words like "19", "8-", "-4", "7.", ".1", "5' ", " '8", "3.2", "2/7", "23%", all the single character words, all the words with single "(" or ")" and all the words with unmatched " " ". By doing this, this Aggregate Analysis Engine attains the precision and F-1 measure ***higher than LingPipe by average 2% and 0.02***, respectively. The precision, recall and F-1 measure distribution table will be shown later.

1.2.4 Word Annotator II (AbnerAnnotator)

The third annotator – AbnerAnnotator collaborates with the former annotator based on LingPipe to generate potential better prediction result and prevent overfitting. This annotator sets the confidence from AbnerAnnotator as 0 or 1. It follows the strategies below:

For words selected both by LingPipe and Abner, this annotator set their confidence from Abner as 1. In the final output step, this value will be combined with weight parameter AbnerConf [0, 1] to contribute to the total confidence. Words fulfills this requirement will be selected:

$$1 * AbnerConf + confidenceLingpipe > threshold$$

The parameter threshold is also adjustable between 0 and 2.

For words only selected by Abner, this annotator excludes these words. There is uncertainty and information missing when I estimate confidence from Abner. In the former pre-test, Abner also showed comparatively poor performance, so I adopt this strategy.

Below is the precision, recall and F-1 measure distribution in the rectangular area **[AbnerConf * threshold]**:

	0	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1	1.1	1.2	1.3	1.4	1.5	1.6	1.7	1.8	1.9	2
0	5;100;9			74;88;81	78;86;82	80;84;82	83;82;82	86;79;82	88;75;81	92;68;78											
0.1	5;100;9	43;95;59					82;83;83			90;72;80	93;47;63										
0.2	5;100;9	43;95;59	44;93;60				81;84;82	83;82;82		88;75;81		93;47;63	Searching in these grey areas is meaningless.								
0.3	5;100;9	43;95;59	44;93;60	45;92;60			79;85;82			87;76;81											
0.4	5;100;9	43;95;59	44;93;60	45;92;60	45;91;61		77;86;81	81;84;82		86;78;81											
0.5	5;100;9	43;95;59	44;93;60	45;92;60	45;91;61	46;90;61	74;87;80	78;85;81	81;82;82	84;79;81	84;57;68		88;56;67		93;47;63						
0.6	5;100;9	43;95;59	44;93;60	45;92;60	45;91;61	46;90;61	46;89;61	75;86;80		83;80;81											
0.7	5;100;9	43;95;59	44;93;60	45;92;60	45;91;61	46;90;61	46;89;61	46;88;61	76;84;80	80;81;80											
0.8	5;100;9	43;95;59	44;93;60	45;92;60	45;91;61	46;90;61	46;89;61	46;88;61	46;86;60	77;82;79											
0.9	5;100;9	43;95;59	44;93;60	45;92;60	45;91;61	46;90;61	46;89;61	46;88;61	46;86;60	46;84;60	73;61;66										
1	5;100;9	43;95;59	44;93;60	45;92;60	45;91;61	46;90;61	46;89;61	46;88;61	46;86;60	46;84;60	39;63;48	73;61;66	78;60;67	81;59;68	82;58;68	84;57;68	86;55;67			93;47;63	
	row – AbnerConf [0, 1]																				
	column – threshold [0, 2)						$1 * AbnerConf + confidenceLingpipe > threshold$														

P;R;F denotes Precision(%), Recall(%) and F-1 measure(e-2), for example, 82;82;82 means 82% precision, 82% recall and 0.82 F-1 measure.

My analysis for this distribution table:

Yellow Grids: In these grids, $AbnerConf = threshold$, so all the words selected by both Abner and LingPipe will be output, no matter what their real confidence is. This definitely causes a low precision. Large amount of output also

contributes to the high recall.

Green Grids: For the green ones, there is a performance "jump" between the yellow and the green, especially in the precision aspect. For each green grid, $threshold - AbnerConf = 0.1$. This will exclude lots of words with confidence from LingPipe < 0.1 and remarkably increase the precision.

Blue Grids: For these blue grids, they have decent performance. They fulfill threshold – AbnerConf > 0.2. This means that more words with higher confidence from LingPipe are chosen and the weight of AnberConf is comparatively low. There is a high chance that overfitting caused by LingPipe itself exitsts. Base on this, I won't choose parameter pair in this area.

Red Grids: These reds grids are the extreme version of blue grids: no confidence from Abner is involved in the decision process. Overfitting definitely exists in red grids.

Based on the analysis above, I chose **(0.5, 0.9)** as my final choice.

1.2.5 Cas Consumer

The Cas Consumer would do three things:

First, by extracting the confidence of LingPipe and Abner and adjusted parameters, it makes the final decision.

Second, Cas Consumer writes the result into a txt file. In this process, it extracts information from NameTag annotation, including id, beginPos, endPos and names.

Third, with function ***PrecisionRecallCalculator***, Cas Consumer outputs the Precision, Recall and F-1 measure.

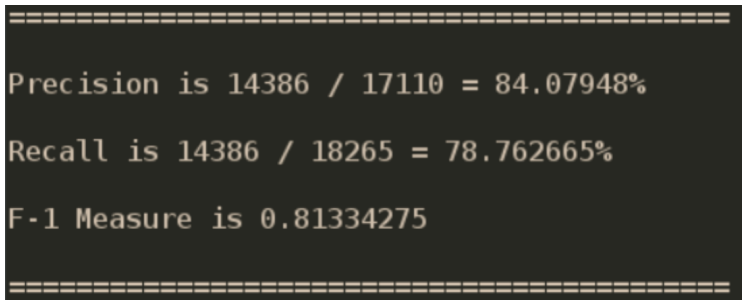
2. Performance Evaluate

2.1 Best performance for given dataset (set AbnerConf as 0.1, threshold as 0.6):

```
==========================================

Precision is 15175 / 18517 = 81.95172%

Recall is 15175 / 18265 = 83.0824%

F-1 Measure is 0.8251319

==========================================
```

In this result, overfitting must exist, so I didn't adopt this.

2.2 My final choice and outcome (set AbnerConf as 0.5, threshold as 0.9):



3. Algorithm Design and External Resources

As we all known, the precision of method using PosTagNamedEntityRecognizer.java from Stanford is intolerably low, so I continue choosing the beloved LingPipe, taking the risk of overfitting. However, I think the overfitting can be avoided to some degree by combining Abner into this project. I find the more percentage Abner involves in the decision, the less F-1 measure I would get. Based on observing the difference between performance of independent LingPipe algorithm and combined algorithm, it indicates the overfitting caused by LingPipe is reduced. There is a trade-off between overfitting prevention and predicting performance, so I decide to give confidence from Abner and confidence from LingPipe similar weights.

Here are my solutions, some of them also can be seen in the former P:R:S distribution table (Page 3):

(1). As for the overall performance (precision, recall and F-1 measure), LingPipe did much better than Abner in this dataset.

(2). If I chose other more trustworthy algorithms with higher precision and recall, the overall performance will definitely improve by combining their confidence and making the decision.

(3). Training other reliable dataset to get model is likely to have a better performance and avoid overfitting.

(4). Analyzing output file and using more regular expression is a good way to improve precision.

4. NLP Tools

(1). LingPipe – use model trained by LingPipe along with LingPipe to extract gene name in GeneNameAnnotator.

http://alias-i.com/lingpipe/demos/tutorial/ne/read-me.html

(2). Abner – just use it without training in AbenerAnnotator.

http://pages.cs.wisc.edu/~bsettles/abner/

5. Data Flow

(1). In SentenceCollectionReader, reader reads all content from the specified input file. Store it into CAS.

(2). In SentenceTagAnnotator, bring document text from CAS and split by lines. Split each line to be id and text, store them as Sentence.

(3). In NameTagAnnotator, use LingPipe and rules to extract all words from document and store them as ids and text with begin position, end position and confidence from LingPipe.

(4). In AbnerAnnotator, use LingPipe to extract Gene from Sentence and abandon words with 0 confidence.

(5). In CasConsumer, output all gene names by comparing the total confidence they have received and the threshold.

Oct/9/2014