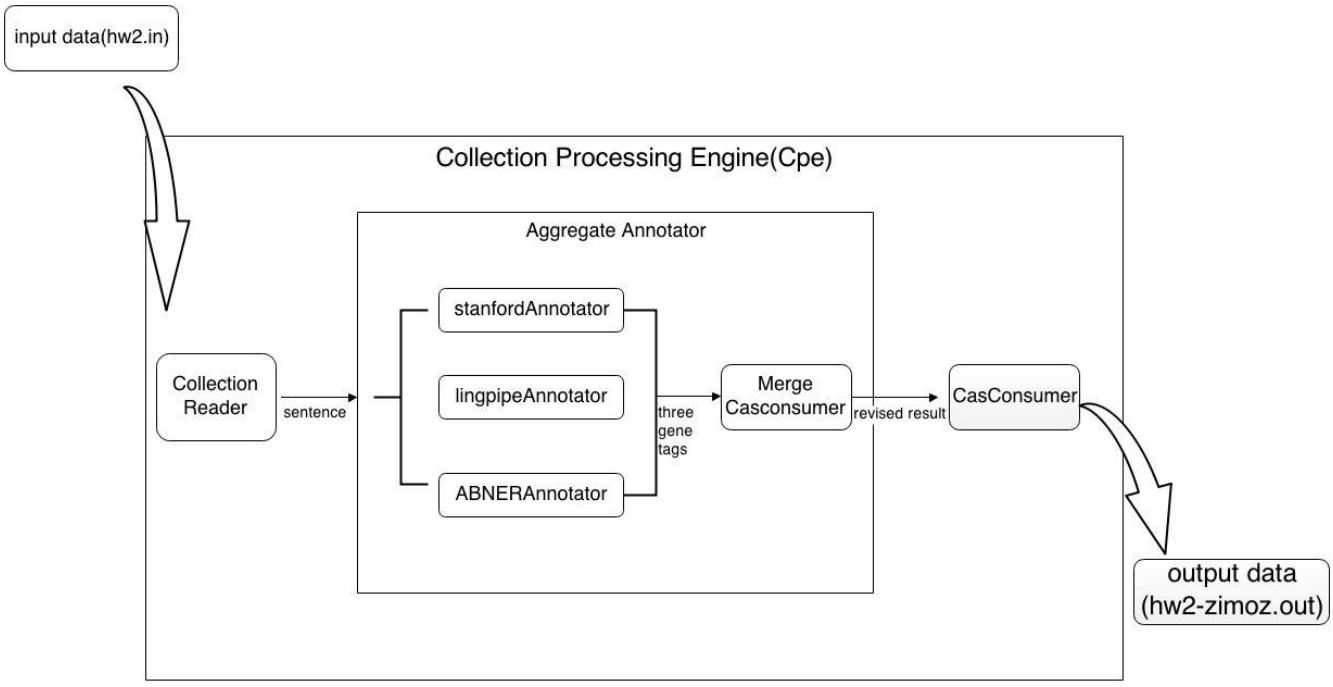
Hw2 Report
Andrew ID: zimoz
Logical Architecture and UIMA Analysis Engines Design & Implementation

1. Introduction

This is the report for homework2. With the background in homework1, in this homework, we are asked to build an aggregate analysis engine, including more than two different gene annotators. As for me, I chose to use three annotators, they are, stanford annotator, lingpipe annotator and ABNER annotator. The report mainly introduces how I designed these three annotators and aggregates these annotators.

2. Design

The following chart is the whole process of my project. It shows how the input texts convert to split ID, positions and gene names.



2.1 Collection Reader

This part is as same as the hw1. The Collection Reader reads the input file by lines and initialize the CAS. It saves the data in string and converts format ready for inputting into Analysis Engine.

2.2 Type System

I set three types in this project. The first two are same with last project. They are geneTag and sentenceTag. I use geneTag to store the gene ID, gene position and gene names and sentenceTag for

storing texts by lines. As for the third type, I set a final result type. Just as its name says, it is set up for storing the final result of gene information we selected from the results of the three different annotators. Also, I pass finalresultTag to CAS Consumer for outputting.

2.3 Annotators

2.3.1 StanfordAnnotator

In this annotator, I just copied the codes from hw1 at first. But for adjusting the format for lingpipeAnnotator which could not initialize the CAS Consumer before, I change the format in this annotator. But the core thought is still same. I just call the stanford-corenlp dependency in Maven, and use its method to recognize the gene names.

2.3.2 LingpipeAnnotator

In this last homework, because of my incidentally delete the eclipse software and had to rebuild the project, I had no time to recover the lingpipe annotator. This time, I rebuild the annotator and add an variable to record the confidence of each gene I find. The function of confidence also prepares for the next aggregate process. Also, this annotator uses another third party resource, and I call the method to extract gene names in the text sentences.
There is a pity that I was about to get the confidence of the results using the get.score() function. But the outputs are all “Infinity”. At that time, I finally found out that only NBest method can get the confidence numbers. I have no time to change my method, so I did not get the confidence. Instead, I use the precision as a measurement in each annotator.

2.3.3 ABNERAnnotator

This is the last Annotator I wrote. Because as I said in homework1, the stanford method has such a low precision, so I do not think it can make sense in improving the output with other annotators like lingpipe. So I tried some new methods, like EntrzeGene. Finally, I chose the ABNER Annotator because of it has good recall and precision, as well as rapid process and easy to implement.
Specifically, I used the BIOCREATIVE corups given in ABNER.

```
Tagger ta = new Tagger(Tagger.BIOCREATIVE);
```

From the Javadoc of ABNER, the result is to return a vector of 2D string arrarys. So I set a 2 dimension array to store the results from this annotator. As the codes show below:

```
String[][] result = ta.getEntities(sText);
```

2.3.4 MergeAnnotator

This part is the key task of this homework. After building up three different annotators, we need to aggregate them and get an AAE replacing the AE position in CpeDescriptor. Because each annotator has one result set, and we have to select and merge the data to get a final result. We need to take advantages of each annotator and improve the final precision, recall and F measurement.
In this part, I spent really long time and had many ideas.(Showed in the Algorithm part below.) Due to the lack of confidence, I chose to use the pure counting methods. I have three annotators and I define the rules that if there any two of annotators vote for a word to be a gene name, we put it into the final

result. It is quite simple and clear. In this way, I got precision of 0.8, which is really good. But the recall is a little disappointing.

2.4 CAS Consumer

Here we comes the last step of the whole system. CAS Consumer generates the result of CpeDescriptor in the format we defined in its java file. In this case, it outputs the final gene selected and merged by three annotators. I also put the evaluation codes in this java file. We can measure the performance of each annotator using primitive AE and find out a merge plan considering performance of each annotator.
Also, in the homework1, lingpipe method is sort of over fitting. So, although we may get a better precision using the lingpipe result directly, combining the results can avoid over fit and get a more practical result.

2.5 Design Pattern

In this system, each class has its responsibilities and fixed positions. Collection Reader is assigned to read the input file and make each sentence a CAS passing to analysis engine. And CAS Consumer is to out put the results from AE in fixed format.
It is quite like the last homework and the most significant feather is that we use the aggregate analysis engine to apply more than one annotators. I use the fixed flow, and let the sequence be “abner-lingpipe-stanford”, which actually does not matter I think in this case.

3 Algorithms

3.1 Annotators

Algorithms for Lingpipe an Stanford have been talking about in the report of homework1.
As for the algorithm of ABNER, the core is a statistical machine learning system using linear-chain conditional random fields (CRFs) with a variety of orthographic and contextual features. It can simultaneously recognize multiple named entities (2 trained models included) and directory-recursive batch annotation of text files. Also, it has optional built-in tokenization and sentence segmentation algorithms, robust to wrapped lines and bio-medical abbreviations.

3.2 Comparison and combination

Because I have three annotators and each one out put a different record, for the aggregate analysis engine, it is the key point to find out a proper way to choose gene name from one of three records.
At first, I was thinking about that simply compare the precision of the annotators and take the highest one as the final output. When testing, I found it ignored the other two annotators and just took the Lingpipe's result. In this way, lots of data did not take into consideration thus greatly reduce the practical use of this system. Then, I came up an idea that I could define the weight of each annotator based on its precision. Reading several papers on this topic, the best way to give the weight number is to dynamically allocate the weights, which means once we make an allocation, we will get a new precision and everytime we get a precision, we reallocate the weights. We can repeat this process until we get convergence. It seems like a really good idea, but needs too much work to implement it in such

a limited time. I recorded this method, and planed to try it later.

4 Evaluation

4.1 precision and recall

For each annotator alone:
Abner precision=0.7643501536631307
Lingpipe precision=0.7648884291414216
Stanford precision=0.10189473684210526

For the final result after merging:
precision=0.8021737732330053
recall=0.6828907747057213
F-measure=0.7377417637664871
correct=12473
total recognizations=15549
Completed 15000 documents
Total Time Elapsed: 144282 ms
Initialization Time: 1328 ms
Processing Time: 142954 ms

We did not get a “much better” result than the formals ones. But for the precision as high as this case, it is hard to improve the precision feather like normal. In addition, the recall rate is a little bit disappointed. I think about there a really low precision in stanford methods, so according to the algorithm in mergement, it may influence the other two annotator results and stop some “correct” from outputting. Anyway, I still think that dynamic weight’s allocation is most reasonable. If still have time, I will at lease put precision into consideration.

4.2 final result

Here is the performance report for this project.

---------- PERFORMANCE REPORT ----------

Component Name: collectionReaderDescriptor
Event Type: Process
Duration: 431ms (0.3%)
Result: success
Component Name: hw2-zimoz-aae
Event Type: Analysis
Duration: 141541ms (98.77%)
Sub-events:
    Component Name: aeDescriptorMerge
    Event Type: Analysis
    Duration: 378ms (0.26%)

Component Name: aeDescriptorLingPipe
Event Type: Analysis
Duration: 6241ms (4.36%)

Component Name: aeDescriptorABNER
Event Type: Analysis
Duration: 96059ms (67.03%)

Component Name: aeDescriptorStanford
Event Type: Analysis
Duration: 38115ms (26.6%)

Component Name: Fixed Flow Controller
Event Type: Analysis
Duration: 462ms (0.32%)

Component Name: hw2-zimoz-aae
Event Type: End of Batch
Duration: 76ms (0.05%)
Component Name: casConsumerDescriptor
Event Type: Analysis
Duration: 1204ms (0.84%)
Component Name: casConsumerDescriptor
Event Type: End of Batch
Duration: 45ms (0.03%)

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. Stanford-NER API
http://nlp.stanford.edu/software/crf-faq.shtml#extend
7. Abner API
http://pages.cs.wisc.edu/~bsettles/abner/javadoc/