Homework2

kangh

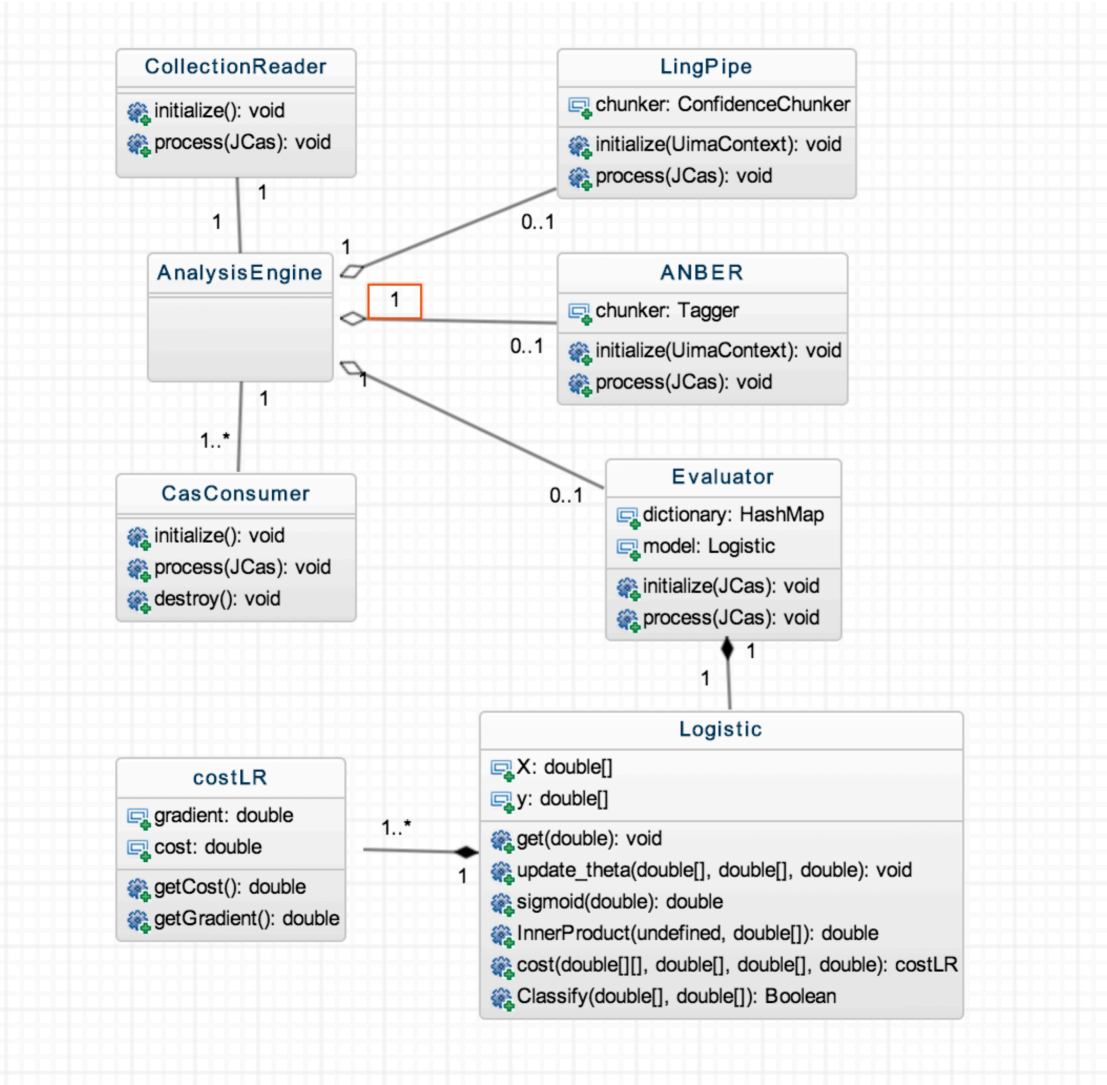
Preface: In homework1, I have already used LingPipe to identify GeneTag. In this homework, I put effort on bringing in new NER tools Abner and make LingPipe and Abner wrapped by AAE descriptors. I Particularly explore how to use Logistic Regression to merge result from results of two NER tools.

This project implements a named entity recognizer based on UIMA SDK filtering out gene-related words plus its position in text.

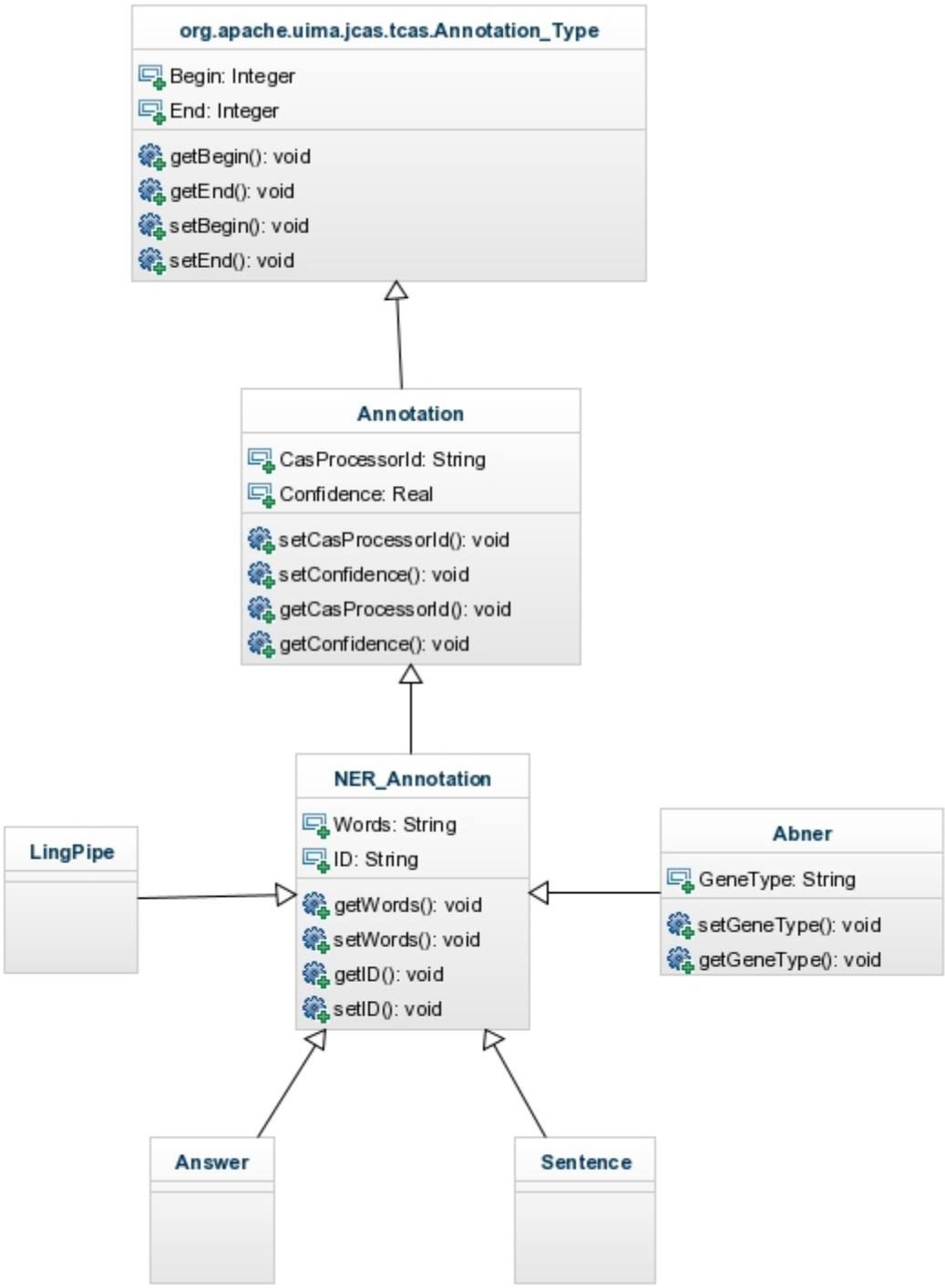
UML:

To clearly demonstrate architecture, here gives two class diagram on and UIMA GenTagNER System and Type system.

UIMA GenTagRecognizer :



Type System:

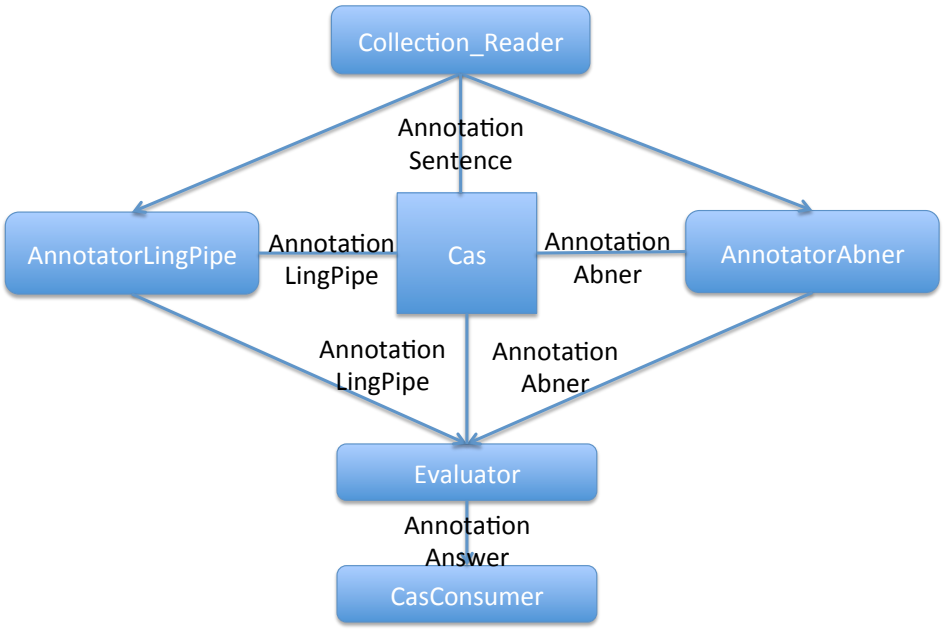


There is a hierarchy Type system. First, Base Annotation inherits from UIMA annotation that has attributes Begin and End. It also add new fields: CasProcessId and Confidence. Then NER Annotation inherits from Annotation by adding Words and ID. Finally, NER Annotation derives many other useful types: Sentence, Abner, LingPipe, and Answer.

The whole projects can be divided into two main parts: Architecture and Algorithm. The architecture is based on UIMA pipeline. The Figure 3 demonstrates the basic flow chart.

Flow chart:

Figure 3



In this flow chart, the Collection Reader read one document sentence by sentence; it will save the result into Sentence Annotation. Then, LingPipe Annotator and Abner Annotator read sentence annotation from Cas almost at the same time. It will analyze the sentence extracting potential Gene-related words into correspond LingPipe Annotation and Abner Annotation. The Evaluator will merge these two results and store final result in Answer Annotation. CasConsumer will pinpoint the position of each word.

Algorithms About Merging Results:

Fact:

Through the test on Golden-Standard answer, I found that F1- Measure of LingPipe using Confidence Named Entity Chunking can reach 0.8, while Abner based on NLPBA model can only have 0.5.

Two naïve solutions:

We first run these two annotators independently, according to the figure 3, the Annotator LingPipe and Annotator Abner read the sentence annotation from Cas on parallel. Then Evaluator will retrieve these two annotations above for merging the results.

Generally, there are two quite straightforward methods:

1. Union the result of LingPipe and Abner by filtering out duplicates.

This solution can greatly guarantee the high recall in sacrifice of precision. We just build a HashMap<words, LingPipe> to store all the results from LingPipe annotation. Then we iterate result of Annotation Abner, to see if it is redundant based on both words and offset in original sentence. However, because of performance difference, Abner will drag down the integrate F1-Measure, which means improving little recall by losing much precision.

Experiment results:

Precision: 0.2746231243 Recall: 0.9391231231

F1-Measure: 0.4249733858630845

2. Based on confidence of LingPipe, use Abner to recheck words below confidence threshold.

The confidence threshold is the key to second solution. Confidence Named Entity Chunking of LingPipe has a very nice function, which can return confidence of some words considered to be Gene-related words. So, assuming that we manually set a fixed threshold that the words with confidence higher than threshold will be directly sent to answer annotation. Otherwise, let Abner check whether such a word negated by LingPipe will be a potential Gene-related word. Unfortunately, there is a hard issue that how to set this confidence threshold, during my programming, I just pick some value intuitively to test its performance. So it is very hard to set convincing threshold without running experiment.

Experiment result:

Threshold	0.65	0.5	0.3	0.15
Precision	0.78321312	0.642342223	0.43244232	0.28312313
Recall	0.83931232	0.874234342	0.89313112	0.91312312
F1-Measure	0.81029289	0.740559551	0.58273299	0.43222912

Analysis: As we can see the F1-meansure drops down quickly as we set low confidence threshold. This is because LingPipe has more competitive effect on final result contrasting to Abner.

Summary: In real experiment, I found that second solution has more flexibility because it can make tradeoff between precision and recall by adjust confidence threshold. The first solution may give away to second solution due to its very low precision dependent on real demand for precision and recall.

However, both solutions have one common potential trouble of relying on LingPipe. LingPipe and Abner has different Gene library. If LingPipe model failed on some situation without much correction by Abner the result will be even worse. But they have both equal rights to vote for Gene words, the average performance will be lower. So can we find a better way to prevent worst case while guarantee acceptable performance?

Machine Learning technique: Logistic Regression

Confidence, as a metric, reflects the possibility of the words could be a Gene-related word. So if we both have confidence from LingPipe and Abner, we can use machine learning technique to find some pattern hiding in data. The machine learning algorithm I choose is Logistic Regression. There are several reasons: First, Logistic Regression is easy to realize. Second, it is a very effective binary classifier wildly used in industry corresponds with our requirement of task. So, data and I capsulate all function structure in Logisitic.java, I can adjust the number of parameters without modifying codes. I test code in LogisticExample.java using rule of Majority of binary string (given a binary string, if there are more zeros output 0, else output 1).

There are two features: confidence of LingPipe, confidence of Abner. LingPipe will generate words embarrassingly low confidence below 5%. We need to remove these parts of words. So there is a initial threshold for LingPipe to send

possible words to next evaluator. Also, I know that only LingPipe has confidence while Abner don't have, so I go to the official website finding a report about its F1-Measure is 0.7. So I approximately set 0.7 on Abner's confidence. We manually add one column 1 into input dataset. It cost 200-300iterations to converge. I run another CpeDecriptor.xml to derive parameter "Theta" of LR weight. So it doesn't bother to train the model in real time, which saves much time. Actually, I only need function "Classify" of Logistic.java, but to prove that I done it independently, I put both Logistic.java and LogisticExample.java into my project. The result is quite exciting:

We adjust confidence threshold to training different binary classifier. Here is the F1-Measure on training dataset.

Confidence	0.05	0.1	0.15	0.2
Precision	0.61742274	0.622832956	0.62412313	0. 62876214
Recall	0.598357514	0.588119353	0.58163123	0. 57760744
F1-Measure	0.607740643	0.604978598	0.60212845	0. 60210021

We see that although whole process has been weakened by almost 20%, but its result is still acceptable, most importantly, when the worst case appears, Abner's influence will be much bigger.

Interesting Findings:

In Evaluation part, I used logistic regression to merge two results. However, I discovered that, actually my features are much less than my dataset, two features versus nearly 15000 words. I think there are two basic way to further improve performance of my LR classifier. First is to add more annotator to screen out potential Gene-related words. In this way we can have more features. Second is to use Kernels to map my low dimensions to higher dimensions. Due to time limit, I have no time finding extra library. So I decide to use Kernels, but the result becomes worse, it is possibly that I make wrong derivations. It will be my next step to make my GenTagNER system robust.

Reference:

1. Ling Pipe, Confidence Named Entity Chunking, website:

http://aliasi.com/lingpipe/demos/tutorial/ne/read-me.html, 2014, 10, 10