IIS — hw2

Chu–Cheng Lin `chuchenl`

October 2014

1 System Overview

The system is very similiar to Homework 1. In fact, the only major changes are the consumer and the addition of another Analytic Engine.

In this homework, we use two Analytic Engines. One uses LingPipe's GENETAG tagger to recognize named entity chunks with confidence scores. The other one uses a regular–expression based genomics chunker (GENIA), which seems to be deterministic, and is also from Lingpipe. We then combine results from the two chunkers together to get our final results.

However, blindly taking all recognized chunks yields poor precision. The reason is that the GENIA chunker is not very good and produces a lot of false positives. So I take the weighted sum of the two chunkers' confidence scores. For a term x, the re–weighted score $L(x)$ is

$$L(x) = w_T c_T + w_G g(x), \tag{1}$$

where $g(x) = 1$ if the GENIA chunker recognizes x; otherwise $g(x) = 0$. We then only keep terms $\{x|L(x) > \theta\}$. This gives us three free parameters w_T, w_G and θ. Since L is just a linear combination, we fix $w_T = 1$ and do a grid search for w_G and θ that maximizes the F–Measure of *sample.in*. Since the GENIA chunker is way too noisy, we just consider terms that the GENETAG chunker finds. We search for $0 \leq w_G \leq 2$ and $-4 \leq \theta \leq 0$ with step size 0.1. The best configuration we have found is $w_G = 0.3$ and $\theta = -0.6$. The best configuration gives a F–Measure of 0.81, slightly better than what we get from the GENETAG only (F–Measure = 0.80).

2 Details

The singleton design pattern is applied for both `GeniaChunker` and `GenetagChunker`, to make sure only one model file is ever loaded. The rest is pretty much like the last homework.

In this homework, we use an aggregate engine instead of a primitive one. The aggregate engine runs the two annotators `GenetagAnnotator` and `GeniaAnnotator` in a fixed flow. We still use the `GeneAnnotation` type; but this time it inherits

from the superclass `edu.cmu.deiis.types.Annotation`, and has a confidence property. We also include a source property to allow identification of source in the consumer.

The consumer `MyConsumer` loads parameters from the configuration file. The consumer has two modes. One mode is search. In this mode, the parameters `(Genia|Thres)(Upper|Lower)` are used to decide the search range. Predictions of all configurations are then written to separate files to allow offline inspection. There are also parameters `Thres` and `Genia` which store the best θ and w_G respectively. To switch between modes, set the boolean parameter `Search`.

Searching for the best parameters is quite straightforward. We have a Python script that evaluates every configuration, and prints the best one.