11791 - Homework 2 - Logical Architecture and UIMA Analysis Engines Design and Implementation

Mohammad Gowayyed
mgowayye@andrew.cmu.edu

October 10, 2014

Abstract

In this document, I report briefly my implementation for the logical architecture of the gene named entity recognition. I used combined two approaches: 1) simple dictionary with names of genes from an online database. 2) (hw1)I used the algorithm described in [1] and used some of the published implementation at [2]. Whenever I use any of their code, I am clearly stating this within my Java code. I use CRF from the Mallet library after extracting token features.

1 System Architecture

1.1 Implementing aggregate analysis engines

I implemented two aggregate analysis engines. First one: CRF-Model that holds all annotators related with the CRF model. The second one: hw2-mgowayye-aae that holds everything.

1.2 CRF Model

The addition in this homework is that this model creates a new type of annotation called PredictedGene that inherits from the Annotation you provided and has a confidence and processor ID.

Figure 1.1 shows the basic architecture of the system. As in the UIMA tutorial, it is recommended to use Analysis Engines instead of CAS Consumers. I have a collection reader that reads the document. Then a preprocessing annotator that creates Sentence objects. Then the feature extraction annotator, that do tokenization, POS tagging and several other features as described in the next section. Based on whether we run a training or a testing program, we will use either training or testing Analysis Engines.

The Banner framework uses Mallet (a machine learning library) for training the CRF. They also extract features within the learning process (using Mallet pipes), which I found not a good design, as it depends (high coupling) on the machine learning library used. I preferred to separate the feature extraction from the learning totally.

Benefits of separating the features extraction from the learning:

1. It reduces coupling between feature extraction and the machine learning library used.
2. It is more modular, as we can replace the feature extractor with another feature extractor without needing to change the learning algorithm. Also, we can use a completely other library for learning or another algorithm from the same library without worrying to edit the feature extraction annotator.
3. It is more reusable, as we can use the same exact annotator for both training and testing.

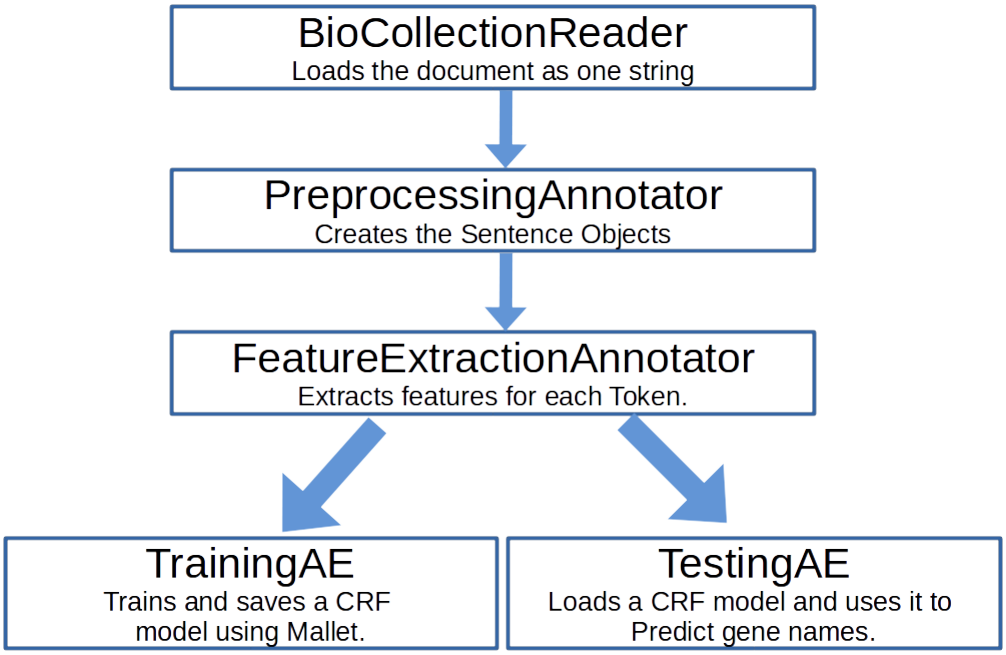

Figure 1: The general architecture of the system.

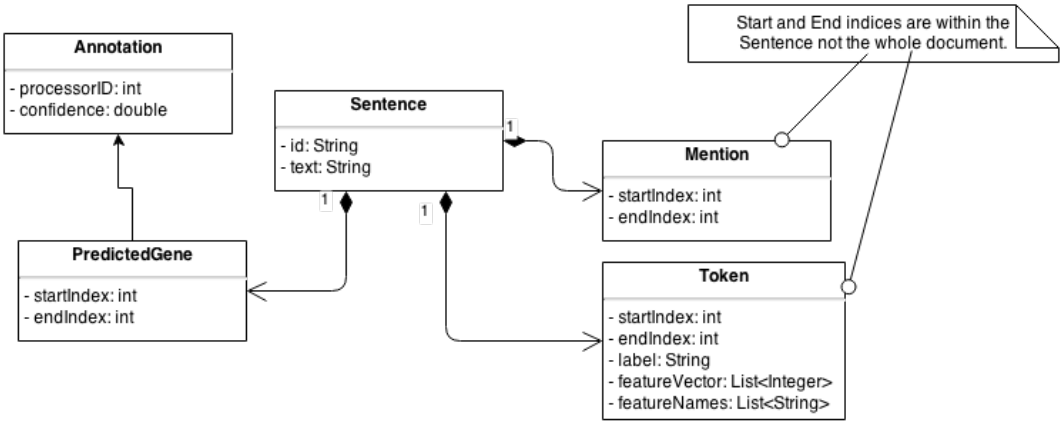

Figure 2: A simple class diagram for the Type System I used. The sentence represents a line of the input and the Token is a token in the Sentence. A Mention represents a gene mention.

1.3 Dictionary Model

This annotator is responsible for searching a dictionary of gene names for the gene and create PredictedGene annotations.

1.4 Combining the two models

I combine the two models by running a Final annotator that aggregates the genes detected by both annotators.

2 Data Flow and Algorithm

2.1 Data Flow

2.1.1 BioCollectionReader

This collection reader loads a whole text document and stores its contents in the JCas document text.
Input: The path of the file to load the data.
Output: One JCas object with the whole document text.

2.1.2 PreprocessingAnnotator

This annotator is responsible for creating Sentence objects from the document loaded by BioCollectionReader.
Input: JCas with one String representing the whole document.

Output: List of Sentence objects, each has an id and text.

2.1.3 FeatureExtractorAnnotator

This annotator is responsible for extracting the features for each token of the Sentence. It uses StanfordCoreNLP classes to 1) tokenize 2) get POS tags. It also extracts other features the same as BANNER [2] system does: Regular Expression features and other linguistic features.

This annotator is generic to work with either training or testing. It is also independent from the learning algorithm and independent from the learning library. In contrast to the BANNER library in which both feature extraction and learning are done using the Mallet library, I separated feature extraction from learning for a better design, so that I have the ability to compare the performance when using different learning algorithms or even different libraries without needing to change my feature extraction annotator.
Input: JCas with List of Sentence objects.
Output: List of Sentence objects, each has list of tokens and each token has list of feature values.

2.1.4 TrainingAnnotator

This analysis engine is responsible for training a CRF model using Mallet library and stores the model to a file.
Input: JCas with List of Sentence objects, each has list of tokens and each token has list of feature values.
Output: CRF model file trained using the features extracted in the previous step.

2.1.5 TestingAnnotator

This analysis engine is responsible for loading a CRF model and using it to predict gene names, then evaluate the predictions and saves them to an output file.
Input: JCas with List of Sentence objects, each has list of tokens and each token has list of feature values.
Output: Detected gene names in an output file and F1-score calculated and printed to the console.

2.1.6 DictionaryAnnotator

This annotator is responsible for searching a dictionary of gene names for the gene and create PredictedGene annotations. **Input:** JCas with List of Sentence

objects, each has list of tokens and each token has list of feature values.
Output: Add PredictedGene objects to the sentences that found on them.

2.1.7 FinalGenePredictorAnnotator

This analysis engine is responsible for combining the CRF Model with the dictionary model and writing the final output file.

2.2 Algorithm

BANNER [1] uses Conditional Random Fields (CRF), which assigns a probability distribution for the labels of each token. First the sentence text is tokenized, then for each token they extract features: POS, Lemmatization, and several RegEx features[1]. Then they train a CRF model using these features.

For testing the CRF model provides the label that has the highest probability for each Token.

3 Experiments

I used the *sample.in* and *sample.out* files to train and test the model. I gradually added features to make sure that more features is reflected into an increase in the F1-score.

When training using only the words as features, I got **0.366** F1-score. Adding POS tagger increased the F1-score to become **0.702**. After adding regular expression features (find their details in ae.FeatureExtractorAnnotator::addRegExFeatures), my final F1-score became **0.893**.

References

[1] BANNER: an executable survey of advances in biomedical named entity recognition, Leaman, Robert and Gonzalez, Graciela and others, Pacific Symposium on Biocomputing, 2008

[2] http://banner.sourceforge.net/

[1] find their details in ae.FeatureExtractorAnnotator::addRegExFeatures