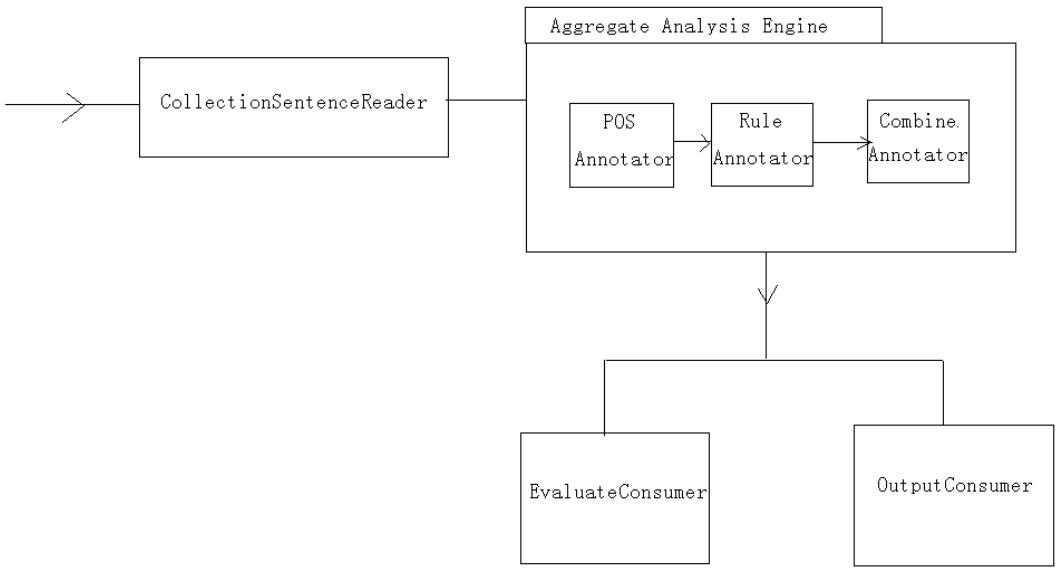
By Yipei Wang

Contact: yipeiw@andrew.cmu.edu

Framework

The system supports two pipeline, prediction pipeline and evaluation pipeline. In prediction pipeline, it reads input file including sentences to be annotated and finally outputs the annotation result to the output file in certain format. In evaluation pipeline, the only difference is that we compare the annotation result with answer instead of output the result.

The named entity recognition is implemented by an aggregate analysis engine. It includes 2 different annotators to recognize the Gen using different methods. The CombineAnnotator is responsible for combining the the annotation generated by each annotator and generate the final annotation.



TypeSystem

Tagger.Gen
--sentenceID --StartOffset --source --confidence
Tagger.CombineGen
--sentenceID

--StartOffset

--source

--confidence

We define 2 typesystems. POSAnnotator and RuleAnnotator can output Tagger.Gen type. The CombineAnnotator use Tagger.Gen as input type and use Tagger.CombineGen type when generating combined annotation. Tagger.CombineGen is the input type of CasConsumers.

The 2 types have same features.

--sentenceID It identifies the sentence including the Gen entity.

-- StartOffset We have certain requirement of the range of the Gen entity and it's different from the word range in the document. Thus, we add StartOffset feature to record the position information we need.

--source It stores information of which annotator generate the annotation. We need to know this information when combine result. Besides, it's useful when compare performance of different annotators.

--confidence It stores how confident is the annotation. Since we use rule-based method, we didn't assigned value to confidence feature now. But it can provide useful information when combining annotation and compare performance.

Components Description

CollectionSentenceReader

It's responsible to read the input file and return CASes that contain the sentences to analyze. It opens the input file when initialized. Then, it generates one CAS for one sentence until processed all sentences in the document.

Why generating CAS for sentence instead of for the whole document?

The most significant reason is that we might get into trouble when the document is large. If we generate CAS for the document, we have to load the whole document. The buffer would be out of memory when loading a large file. And we might generate a large amount of annotations over all the sentences. This would also lead to out of memory problems.

Also, it's convenient to get the range information and sentence identifier by processing each sentence. The annotators only need to consider the annotation problem in a sentence. They don't need to process the task of segmenting document.

POSAnnotator

It uses syntactical rules to do the annotation. It's responsible for transforming the result of the tool into the required annotation and adds source information.

RuleAnnotator

It uses the orthographical rules to do the annotation. It's responsible for transforming the result of the tool into the required annotation and adds source information.

CombineAnnotator

It reads the annotation by other annotators and generates combined result. We support different combine strategy. You can choose annotation only from one source or the intersection of annotations from different annotators.

OutputConsumer

It outputs the annotation from the aggregate analysis engine in a required format.

EvalConsumer

It performs the evaluation task by comparing our annotation and the labeled file. It loads the true labels when the first called. Then, it compares the annotation in CAS with this answer set each time and update the statistics. It supports the evaluation measure of precision and recall.

NLP technologies

We use rule-based method. The rules cover two sources of information, syntactical and orthographical. We tokenize the sentence using Stanford nlp tool and apply the rules on each token.

Rules sets:

Syntactical:

- Part-of-Speech

  If the Part-of-Speech is noun, it's a Gen entity.

Orthographical:

- Length of word

  If the token is a single word and the word is longer than the threshold, it's a Gen entity.

- Capital character

  If there exists capital character except for the first character in any word of the token, it's a Gen entity.

- Special character

If there exists special character, like '-','(', etc. in any word of the token, it's a Gen entity.

- Digits

  If

If any of the word in the token is consisted letter and digits, it's Gen entity.

Design pattern used

Low coupling

Utility class

The CountNoSpaceChar function is used by different annotators. So we separate it and put it in Utility class. This increase the reusability of code.

POSTaggerNamedEntityRecognizer class & RuleTaggerNamedEntityRecognizer class

We didn't implement the recognized method in the annotator. Instead, we implement the method in other class. This leads to the high cohesion of each class.

And in RuleTaggerNamedEntityRecognizer class, each rule is implemented in one function, this is also low coupling pattern. We can flexibly combine different rules in the recognizer latter.

OutputConsumer class

There is a separate output function which supports the required format. This allows us to support different format if needed.

Singleton

In the EvalConsumer, we only load the true label file once and stored it in the memory. This reduce the cost of loading answers when we compare the annotations from each CAS.

Experiment Result

We conducted experiment on the data of sample.in and sample.out. We compared the performance of using only syntactical information, only orthographical information and the intersection of the2 information. Since the rule is quite simple and we didn't make use of any context information, the performance is not very good.

Only using Part-of-Speech leads to high recall but low precision. The value of recall is over 0.9 and the precision value is about 0.09. This is easy to be understood. Because all the Gens belong to noun phrases.

Only using orthographical information leads to much lower recall and a little higher precision. The value of recall is about 0.6 and the precision value is near to 0.1.

When took the intersection of the 2 information, we didn’t get any improvement. It’s mainly because the phrases bearing the orthographical rules are basically noun phrases. So the intersection didn’t add new information to recognize the Gen entity.