Logical Data Model and UIMA Type System Design & Implementation

Name: Xiang Li Andrew ID: xiangl2 E-mail: lixiangconan@gmail.com

1. Requirement Analysis

In this task, we need to build an information processing system. The system reads the input text from a file, and generates the answers after processing it. The input file contains a question and several candidate answers to the question (whether an answer is correct has been given by a number in the file). The system needs to choose correct answers from the given candidate answers according to the answers' content rather than the given numbers.

Specifically, the input question starts with an uppercase letter "Q" followed by a sentence, which is the content of the question. For example, the input sentence could be:

Q John loves Mary?

The input file also contains several candidate answers. Each answer has three parts: the first part is an uppercase letter "A" that indicates it is an answer; the second part is a number (1 or 0) showing whether the answer is correct; the third part is a sentence, which is the answer's content. These three parts are separated by spaces. For instance, one input answer could be:

A 0 John doesn't love Mary.

Since the second part is "0", we can know that the answer is actually not correct.

After processing the input file, the system will choose N answers from the candidate ones as the correct answers. Then a precision that measures the performance of the system will be calculated according to how many answers given by the system is actually correct.

2. Processing Pipeline

To achieve the final goal, the system will use a multi-step pipeline to process the input file. The pipeline is shown in Figure 1:

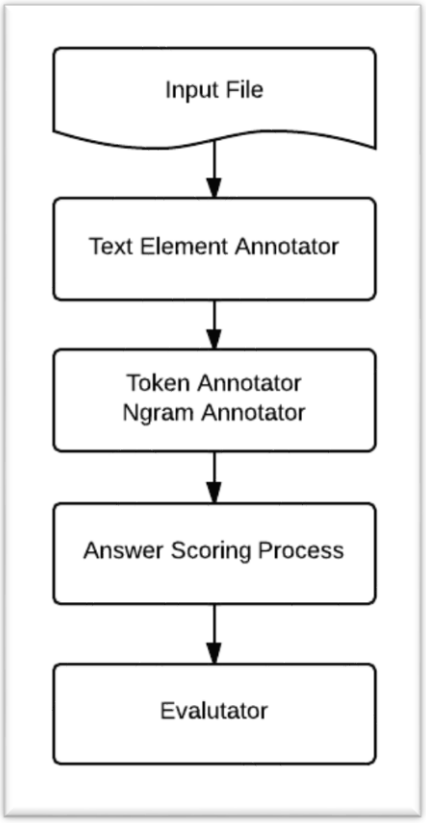

Figure 1 System Pipeline

According to the pipeline, the input file is firstly processed by the Text Element Annotator, which marks the question and answers in the file. Then, the input questions and answers are divided into tokens using the Token Annotator, and consequent tokens are connected into several 1-gram, 2-gram, 3-gram sequences by the N-gram Annotator. After that, an Answer Scoring Process will compare the N-gram sequences in the question and answers, and assign a score to each answer. Finally, the Evaluator will rank answers according to their scores, choose the top N answers as correct answers, and then calculate precision at N to evaluate the performance.

3. Data System

To implement the pipeline, a suitable data system should be built for data storage and transfer. We separate the data system using four namespace: base, input, content and output. Base type contains the basic features for a text annotation, and all other types then inherit from this type.

3.1Base type

There is only one base type, `TextAnnotationBase`, under the base namespace. This type extends the UIMA type `Annotation`, which defines two features indicating the begin and end of a span being annotated. `TextAnnotationBase` has two new features: a feature `source` indicating the annotator that annotated this annotation,

and a feature `confidence` indicating how confidence this annotation was. These features are necessary for other annotations.

3.2 Other types

Other types are defined under namespace input, content and output. Types related to Text Element Annotator are defined under namespace input. These types annotate the input elements, such as questions and answers. Types related to the intermediate processing procedure are defined under namespace content. These types are used for the scoring process. Types related to the result of the scoring process and final answers are defined under output namespace, as these types are finally used to output result. Detailed information about these types are shown in figure 2:

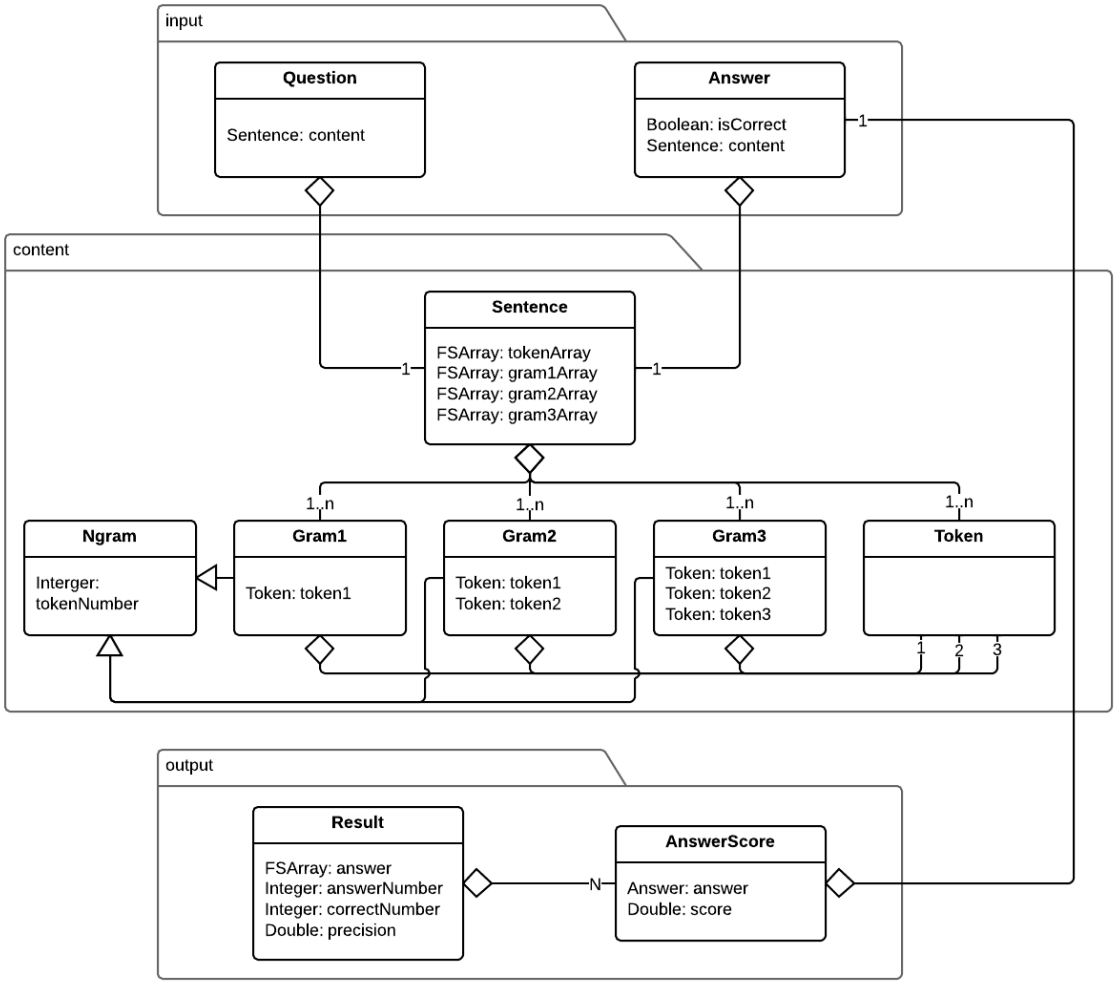

Figure 2 Type System Structure

4. System Explanation

We can explain the type system using a simple example. Suppose the input file contains following text:

Q John loves Mary?
A 1 John loves Mary with all his heart.
A 1 Mary is dearly loved by John.
A 0 Mary doesn't love John.
A 0 John doesn't love Mary.
A 1 John loves Mary.

The system firstly processes the input file using the Text Element Annotator, which generates a `Question` instance for the question "Q John loves Mary?" and five `Answer` instance for the five answers. Then, using the Token Annotator and the Ngram Annotator, the question and answers will be divided into several tokens and N-gram sequences. For example, there should be an `Answer` instance for the answer "A 1 John loves Mary". In this instance, the feature `isCorrect` should be true, and the feature `content` should be a reference of a `Sentence` instance. The `Sentence` instance will stores detailed information about the sentence "John loves Mary" after the sentence being divided into tokens and N-gram sequences. Specifically, the feature array `tokenArray` will store references to the three tokens, "John", "loves" and "Mary", in the sentence. The feature array `token2Gram` will store references to two `Gram2` instance, which are for 2-Gram sequence "John loves" and "loves Mary". Other two feature arrays also store similar things. After that, the Answer Scoring Process will calculate a score for each answer using the tokens and N-gram sequences. For each `Answer` instance, an `AnswerScore` instance will be generated to store its score. Finally, answers with the highest N scores will be chosen as correct answers. The system will also calculate a precision to measure its performance. These results will be saved in a `Result` Instance.

5. Detail Analysis

For the type system, there are many different ways to design its whole structure, as well as specific types. Therefore, we always need compare a variety of plans and find the best one. Here are some examples.

Is it necessary to store references to `Token` and `Ngram` instances in each `Sentence` instance?

In the processing procedure, we will divide the text into several tokens and ngram sequences, which are stored in `Token` and `Ngram` instances. Therefore, we need to concern if it is necessary to store references to them in the `Sentence` type. In fact, the index of these instances are always stored in the system. However, with this index, it is not convenient to compare tokens and ngram sequences in two sentences. So, to speed up the comparison, references to these instances are added to the `Sentence` type. Then, when we need to do the comparison, we can easily find the tokens and ngram sequences in each sentences according to the references and do the judgment.

Do we need to separate the `AnswerScore` type with `Answer` type?

In our system, we separate the `AnswerScore` type with the `Answer` type. The reason for this separation is that we think the `Answer` type is only related to the input text, while the `AnswerScore` type is mainly related to the scoring process. Therefore, in most cases, we do not need to modify the `Answer` type if the input format is fixed. However, features in `AnswerScore` type may change when we modify the scoring process. What' more, there could be more than one scores for an answer if we use multiple scoring method, while these scores may even have different data types. For these reasons, combining the two types into one type may cause many problems. Therefore, we decided to use two types to improve the modifiability of the type system.