11-693 Software Method for Biotechnology Homework 2
Name: Tao Guo
id: taog

Report of Implementing A Simple Information Processing Task with UIMA SDK

1. Requirement analysis
Due to the assignment, the information processing pipeline consists of 5 steps.

1.1 Question and Answer Annotation
In this step, the system reads in the input file, annotates the question and answer spans and record whether the answer is correct or not.

1.2 Token Annotation
In this step, the system annotates each token span from each question and each answer. To enable the possibility to do the scoring efficiently, make connections between the sentence and the tokens.

1.3 NGram Annotation
In this step, the system annotates n-grams of consecutive tokens. The system should also make connections between the sentence and the nGrams.

1.4 Answer Scoring
In this step, the system use gold answer scoring, token overlap scoring and n-Gram overlap scoring to assign scores to each answer, indicating the possibility that the answer is correct to the question.

1.5 Information Processing Evaluation
In this step, the system select the answer with the highest scores as the correct answer, comparing it to the golden answer and calculating the precision in different scoring algorithms.

2. Logical Data Model

The logical data model design is illustrated as the following UML type diagram (figure).

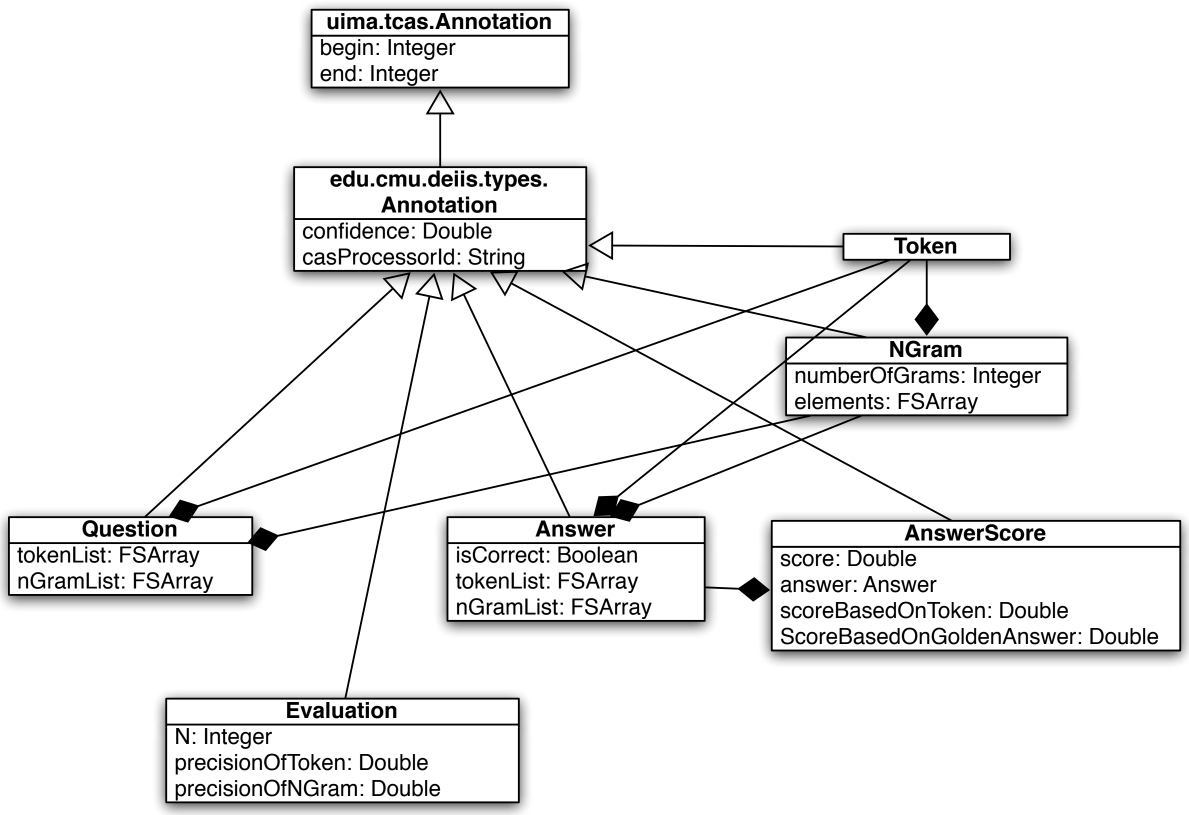

Figure: UML Type Diagram of the Design

2.1 Type *Annotation*

The type *edu.cmu.deiis.types.Annotation* directly inherits from the type UIMA *Annotation*. It inherits the basic characteristics and features such as the *begin* and *end* features. At the same time, it contains the feature *confidence* to help keep track of how confidence the annotation was. And another feature *casProcessorID* to record where it was originally created by. These two features are inherited by all the subtypes.

2.2 Type *Question*

The type *Question* records the question in the input. It also provides *tokenList* and *nGramList* features to record the list of tokens and n-grams, both of them are FSArrays.

2.3 Type *Answer*

The type *Answer* records the answers in the input. It also provides *tokenList* and *nGramList* features to record the list of tokens and n-grams, both of them are FSArrays.

2.4 Type *Token*

The type *Token* denotes to a token in the sentences and the n-grams.

2.5 Type NGram

The type *NGram* denotes an n-gram, thus a contiguous sequence of n tokens from the text. It has a feature *numberOfGram*s to record the length of the n- gram. It also has an FSArray feature *elements* to refer to the tokens it contains.

2.6 Type *AnswerScore*
The type AnswerScore records the scores of an answer using gold answer scoring (feature *scoreBasedOnGoldenAnswer*), token overlap scoring (feature *scoreBasedOnToken*) and n-Gram overlap scoring (feature *score*). Apart from these three score, it also records the original answer annotation using the feature *answer.*

2.7 Type *Evaluation*
The type *Evaluation* records the precision on each scoring system, that is the feature *precisionOfToken* and *precisionOfNGram*. On the other hand, it has a feature *N* that records the sample size that the precision denotes to.

3. Annotator Implementation

In the annotator implementation, the system contains 5 Java Classes. that is *QuestionAnnotator*, *AnswerAnnotator*, *TokenAnnotator*, *AnswerScoreAnnotator* and *EvaluationAnnotator*.

3.1 *QuestionAnnotator*
In this class, the annotator uses regrex to extract the question, creates *Question* annotation and sets the features of it.

3.2 *AnswerAnnotator*
In this class, the annotator uses regrex to extract the answer, determines the golden answer, creates *Answer* annotation and sets the feature of it.

3.3 *TokenAnnotator*
This class is the annotator for **BOTH *Token* and *NGram***. In this class, the annotator splits the string of the annotation *Question* and *Answer* and get the information of the tokens. The class creates these *Token* annotation, sets the feature of it, and set the *TokenList* of the *Question* or *Answer* it belongs to.

After the tokens are created, the annotator combines these tokens together to create the *NGram* annotation, sets the feature of it, and sets the *NGramList* of the *Question* or *Answer* it belongs to.

3.4 *AnswerScoreAnnotator*
In this class, the annotator utilizes the token overlap scoring and n-Gram overlap scoring to give each *Answer* scores. It uses two for-loops to compare the tokens and nGrams of the *Question* and the *Answers*, then generates the scores.

3.5 *EvaluationAnnotator*
In this class, the annotator collects all the *AnswerScore* annotation, sort the score using hash map, determines the number of correct answers, compares the answer with the golden answers and calculates the accuracy of each scoring system. The output is printed to the console.