HW1-wtu-report

On my design of type system for a simple QA system

Name: Wenlong Tu **ID:** wtu

System Description

I will first demonstrate my understanding of this system by concluding the function of this simple information processing system as follows:

1. The input of the system is a document which contains one single question followed by a series of answers to that question. The answers could be right or wrong and are tagged by 0-1 Boolean value just right before their content.
   I assume that only one document will be process in each time the program is executed. Also, I assume that each document has only one question according to the TA in the recitation for hw1.
   *This assumption is of vital importance, because the type system could be quite different if we consider multiple doc/question input. I am not sure whether there would be such demand in the future, but currently my design is based on these assumptions.
2. The output of the system is a sorted answer list based on the scores of these answers. Also, a precision @ N is calculated based on the first N answer's correctness.
3. To produce this output from the input, obviously we need an Answer Scorer to make scores for answers. We can't score raw input text without any pre-processing. As a result, we need several steps to processing the text into computable elements. As it is stated in Task 1.3, we need to have a Test Element Annotation part to parse the input document into question and answers (with their correctness). Then a Token Annotation part and an NGram Annotation part to split sentences into tokens/NGrams which can be used in scoring algorithms. And finally, we need an evaluation part to produce the final output.
   So, the pipe line would be Element Annotation -> Token Annotation -> NGram Annotation -> Answer Scorer -> Evaluation part. The NGram annotation comes after token one is because ngram annotation needs the tokens as input.

Type System Design

*All my type are in the package named *cmu.edu.iis.wtu.model*.

As convention, there is a basic type called *BaseAnnotation* with 2 features: a string

feature *source* and a float feature *confidence*. These features help to track where the annotation was made and with how much confidence.

Name	**Supertype**	
BaseAnnotation	*uima.tcas.Annotation*	
Features	**Name**	**Range**
	source	*uima.cas.String*
	confidence	*uima.cas.Float*

The first part of the pipeline, Test Element Annotation, will split the document into question and answers. Both question and answer are sentences. So I define a type called *Sentence* as the base type of those two annotations. The *Sentence* type has no more features than *BaseAnnotation* for the reason that we only need its content, so *start* and *end* are enough.

Name	**Supertype**
Sentence	*BaseAnnotation*

The type *Question* and *Answer* are extended from *Sentence*. For *Question*, there is no more information to be store or processed, so it basically the exact same as *Sentence*. For *Answer*, I add a Boolean feature called *isCorrect* to identify the correctness gold standard of this answer.

Name	**Supertype**
Question	*Sentence*

Name	**Supertype**	
Answer	*Sentence*	
Features	**Name**	**Range**
	isCorrect	*uima.cas.Boolean*

For the smallest elements, token, there is a type called *Token*. It has no more feature than *BaseAnnotation*, because tokens are simple elements and they only need a begin and an end. Each instance of the *Token* type represents a token that produced by the Token Annotation.

Name	**Supertype**
Token	*BaseAnnotation*

To store the relation between a sentence and its tokens, I define a type named *SentenceWithTokens*. It contains a sentence and the list of its tokens. In the Token Annotation part, the module does not only split sentence into tokens, but also packs them into this type, so that the following part can use this for other processing on the sentence. Also, this type could be used for both question and answer, so there should be a *isAnswer* Boolean tag to identify its category.

Name	**Supertype**
SentenceWithTokens	*BaseAnnotation*

Features	**Name**	**Range**
	sentence	*Sentence*
	tokenList	*uima.cas.FSArray*[*Token*]
	isAnswer	*uima.cas.Boolean*

For the Ngram Annotation part, I define a type named *NGram* to represent the output annotation. The *NGram* contains an array of its token elements. The size n of the *NGram* could be obtain by checking the array length, so there is no need to add a feature to store that.

Name	**Supertype**	
NGram	*BaseAnnotation*	
Features	**Name**	**Range**
	tokens	*uima.cas.FSArray*[*Token*]

For the same reason as *SentenceWithTokens*, I define *SentenceWithNGram*. It's very similar to *SentenceWithTokens*.

Name	**Supertype**	
SentenceWithNGram	*BaseAnnotation*	
Features	**Name**	**Range**
	sentence	*Sentence*
	ngramList	*uima.cas.FSArray*[*NGram*]
	isAnswer	*uima.cas.Boolean*

For the answer scoring, I define S*core* to represent one score made by one scorer for one answer. It contains a String feature called *source* to store the name of the scorer, a float feature called *value* for the score itself and a float feature called *weight* to store the weight of this score in the total score.

Name	**Supertype**	
Score	uima.tcas.TOP	
Features	**Name**	**Range**
	source	*uima.cas.String*
	value	*uima.cas.Float*
	weight	*uima.cas.Float*

For each answer, I define a type called *AnswerWithScore* to represent each answer along with all its scores made by different scorers and also the final score.

Name	**Supertype**	
AnswerWithScore	*BaseAnnotation*	
Features	**Name**	**Range**
	answer	*Answer*
	scores	*uima.cas.FSArray[Score]*
	finalScore	*uima.cas.Float*

Finally, I define a type called *Result* for the evaluation result. It contains the list of all answer sorted by their scores and the final precision of the pipeline.

Name	**Supertype**	
Result	*BaseAnnotation*	
Features	**Name**	**Range**
	sortedAnswer	*uima.cas.FSArray[AnswerWithScore]*
	precision	*uima.cas.Float*

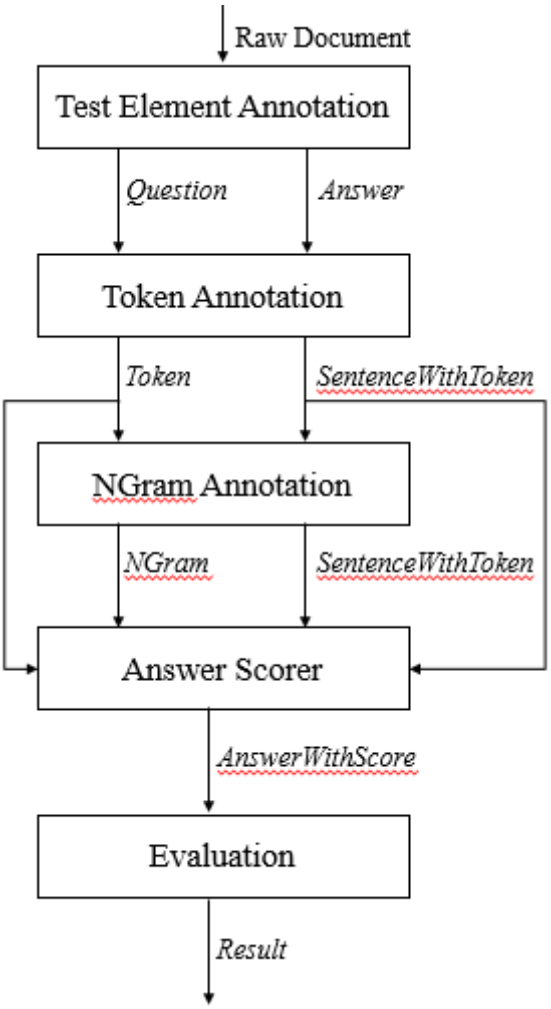
Pipeline Description

The pipeline contains 5 modules, Test Element Annotation, Token Annotation, NGram Annotation, Answer Scorer, Evaluation Module. The input and output of each part can be describe as the following table.

Module	**Input**	**Output**	**Function**
Element Annotation	Input Document	*Question* *Answer*	parse the document into one question and a series of answers
Token Annotation	*Question* *Answer*	*Token* *SentenceWithTokens*	split question and answers into tokens and pack the sentences along with their tokens
NGram Annotation	*SentenceWithTokens*	*NGram* *SentenceWithNGram*	use the list of tokens in each sentence to produce NGrams for each sentence
Answer Scorer	*Token* *SentenceWithTokens* *NGram* *SentenceWithNGram*	*AnswerWithScore*	compare tokens and ngrams in answers with the question to get a score for each answer. *There could be more than one scorers in this module to get a better performance.

Evaluation Module	*AnswerWithScore*	*Result*	sort the answers by scores and calculate the precision.

The diagram of this pipeline would be as follows.



Future Work

For more complex annotation or scorer, we may need to add more details in the result of token annotation and ngram annotation, which means the *SentenceWithTokens* and *SentenceWithNGram* may need more features to store essential information for more complex scorer.

For *AnswerWithScore*, I have considered the situation that multiple scorer could cooperate to generate a better performance. We just need to add multiple score into the score list and assign it with proper weight. However, this approach may only useful for combine scores in linear combination. If we want other way to combine them, we need to achieve that through code implementation.