HW #2 Report: Q&A Annotation Engine

Version: 0.1

Fall 2013 11-791

Jeffrey Gee

Contents

1 Introduction

1.1 Purpose

The purpose of this document is to describe the UIMA Aggregate Analysis Engine for designed as part of Assignment #2.

1.2 Terminology

The following is a list of unobvious terms/acronyms that may be used throughout this document.

Word/Acronym	Definition
AAE	Aggregate Analysis Engine

1.3 Requirements

The following requirements were specified by the Assignment #1 document.

1.3.1 System Input

The input types for this system will be text documents containing a single question statement and a collection of candidate answer statements. All answers will be labeled as to whether they are correct ("1") or not ("0"). An example is shown below.

```
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.
```

1.3.2 Tasks

The following tasks will be accomplished by this system.

1. Assign scores to each answer in regard to the question.
2. Rank the answers according to score.
3. Select the top *N* sentences.
4. Measure performance by how many of the selected sentences are correct.

1.3.3 System Assumptions

The design described in this document is based on some additional assumptions below.

1. Each input document will contain ONLY ONE question.
2. Each answer is relevant to a SINGLE question. For example, answers will not be scored against future questions.
3. Each answer can be scored any number of times. For example, multiple scoring mechanisms can be used to score the same answer against the same question.

4. Tokens and Unigrams are different in that tokens can be transformed (such as stemmed) by a Unigram annotator without modifying the token itself (and therefore not affecting other N-Gram annotators).

2 Analysis Engine

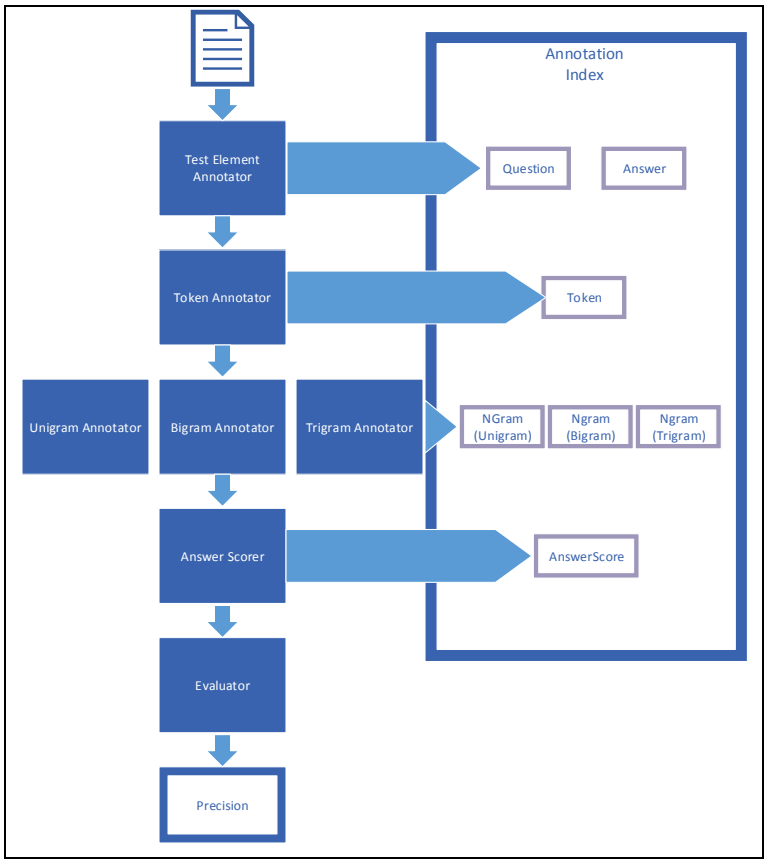
2.1 Introduction

This section will describe in more detail the Aggregate Analysis Engine (AAE) designed in Assignment #2.

2.2 QA System Pipeline

The following is an example of an aggregate analysis engine (from HW#1).

	Annotator	Comment
1	Test Element Annotator	Read in the simple input file and annotate question and answer spans.
2	Token Annotator	Annotate each token span in each question and answer.
3	Unigram Annotator	Annotate each unigram in each question and answer.
4	Bigram Annotator	Annotate each bigram in each question and answer.
5	Trigram Annotator	Annotate each trigram in each question and answer.
6	Answer Scorer	Score each candidate answer against the question
7	Evaluator	Evaluate the precision for a set of AnswerScores produced by a given Answer Scorer.



As can be seen from the diagram above each annotator in the AAE outputs specific annotations corresponding to its role in the chain. In this engine, the AnswerScorer and Evaluator are generic. However, in the next section, we will elaborate further on these components and provide example implementations.

2.3 Abstract Annotators

This section will describe the Abstract Annotators that are extensible in our system.

2.3.1 AbstractQAAnnotator

The AbstractQAAnnotator is the base annotator for this QA system. Rather than having system annotators extend the JCasAnnotator_ImplBase class directly, they should extend this class. The class should store any utility functions that may be used by any annotators in the QA system. For example, a utility function *annotationIsInAnnotation()* is a utility function that returns a Boolean based on whether one input Annotation is contained within the span of another.

2.3.2 NGramAnnotator

The NGramAnnotator is an abstract class for Annotators extracting NGrams. It implements a generic *process()* method, and contains an abstract method called *extractNGramsFromAnnotation()*, which it calls for each Question and Answer span. All extending NGramAnnotators need to implement this method. For example, the Unigram *extractNGramsFromAnnotation()* method produces unigram NGram annotations.

Although it is possible to have a single NGramAnnotator extracting unigrams, bigrams, and trigrams all at once (which is computationally more efficient), there may be cases in which not all the NGrams are needed (such as unigrams). Thus it is more convenient to uncouple each n-gram extractor, and use this abstract NGramAnnotator to reduce code duplication.

2.3.3 AbstractSingleQuestionScorer

The AbstractSingleQuestionScorer is an abstract class that implements *process()* and requires the implementation of a *scoreAnswer()* method. The *process()* method in AbstractSingleQuestionScorer calls the *scoreAnswer()* method for a Question/Answer pair. In the case of this AbstractSingleQuestionScorer, the question is always the same.

2.3.4 AbstractEvaluator

The AbstractEvaluator is an abstract class that implements *process()* and requires the implementation of an *evaluate()* method. The *process()* method calls *evaluate()* on the collection of AnswerScore annotations in the AnnotationIndex. The *evaluate()* method should return a precision value for each collection.

2.4 Annotators

This section will describe the Annotator implementations that make up the AAE.

2.4.1 TestElementAnnotator

Input Annotations	Output Annotations
-	Question, Answer

The TestElementAnnotator reads in input document text and searches for the question and answer patterns described in Section 1.3.1. It adds Question and Answer Annotation objects for detected question and answer spans respectively.

Implemented Methods	Parameters	Comments
process()	**JCas** *Document CAS*	Search for question and answer patterns and add respective Annotations to CAS index.

2.4.2 TokenAnnotator

Input Annotations	Output Annotations
Question, Answer	Token

The TokenAnnotator reads in Question and Answer annotations and outputs Token annotations based on a simple pattern. In this system Tokens are considered to be whitespace delimited chunks of characters. Characters can range from alphanumeric characters to symbols.

Implemented Methods	Parameters	Comments
process()	**JCas** *Document CAS*	Search for token pattern and add Token Annotations to CAS index.

2.4.3 UnigramAnnotator

Input Annotations	Output Annotations
Question, Answer, Token	NGram

The UnigramAnnotator reads in question and answer spans and outputs NGram objects for each token in those spans. The UnigramAnnotator extends the NGramAnnotator abstract class, and does not implement the *process()* method (as this is done by the parent class). It does however, implement the method *extractNGramsFromAnnotation()* method, which iterates through the tokens for each question and answer span and outputs unigram objects.

Implemented Methods	Parameters	Comments
extractNGramsFromAnnotation()	**JCas** *Document CAS*, **Annotation** annotation	Extract all unigrams from an annotation and add them to the document annotation index.

2.4.4 BigramAnnotator

Input Annotations	Output Annotations
Question, Answer, Token	NGram

The BigramAnnotator is identical to the UnigramAnnotator, only that it implements the *extractNGramsFromAnnotation()* method differently, outputting 2-gram NGrams rather than singular-gram ones.

Implemented Methods	Parameters	Comments
extractNGramsFromAnnotation()	**JCas** *Document CAS*, **Annotation** annotation	Extract all bigrams from an annotation and add them to the document annotation index.

2.4.5 TrigramAnnotator

Input Annotations	Output Annotations
Question, Answer, Token	NGram

The BigramAnnotator is identical to the UnigramAnnotator and BigramAnnotator, only that it implements the *extractNGramsFromAnnotation()* method differently, outputting 3-gram NGrams rather than unigrams or bigrams.

Implemented Methods	Parameters	Comments
extractNGramFromAnnotation()	**JCas** *Document CAS*, **Annotation** annotation	Extract all trigrams from an annotation and add them to the document annotation index.

2.4.6 GoldenAnswerScorer

Input Annotations	Output Annotations
Question, Answer	AnswerScore

The GoldenAnswerScorer extends the AbstractSingleQuestionAnswerScorer and implements the *scoreAnswer()* method. The GoldenAnswerScorer scores each answer perfectly, by peeking at its true label.

Implemented Methods	Parameters	Comments
scoreAnswer()	**Question** question, **Answer** answer	Score an answer against a question perfectly, by looking to see whether an answer was labeled correct or incorrect.

2.4.7 TokenOverlapScorer

Input Annotations	Output Annotations
Question, Answer, Token	AnswerScore

The TokenOverlapScorer extends the AbstractSingleQuestionAnswerScorer and implements the *scoreAnswer()* method. Scoring is done by measuring what percentage of the answer tokens can be found in the question tokens.

Implemented Methods	Parameters	Comments
scoreAnswer()	**Question** question, **Answer** answer	Score an answer against a question by dividing the size of the intersection of the answer and question token sets by the size of the answer token set.

2.4.8 NGramOverlapScorer

Input Annotations	Output Annotations
Question, Answer, NGram	AnswerScore

The NGramOverlapScorer extends the AbstractSingleQuestionAnswerScorer and implements the *scoreAnswer()* method. It works identically to the TokenOverlapScorer, only which in the case of the NGramOverlapScorer, the overlap is based on the size of the intersecting set of NGrams.

Implemented Methods	Parameters	Comments
scoreAnswer()	**Question** question, **Answer** answer	Score an answer against a question by dividing the size of the intersection of the answer and question n-gram sets by the size of the answer n-gram set.

2.4.9 TopNEvaluator

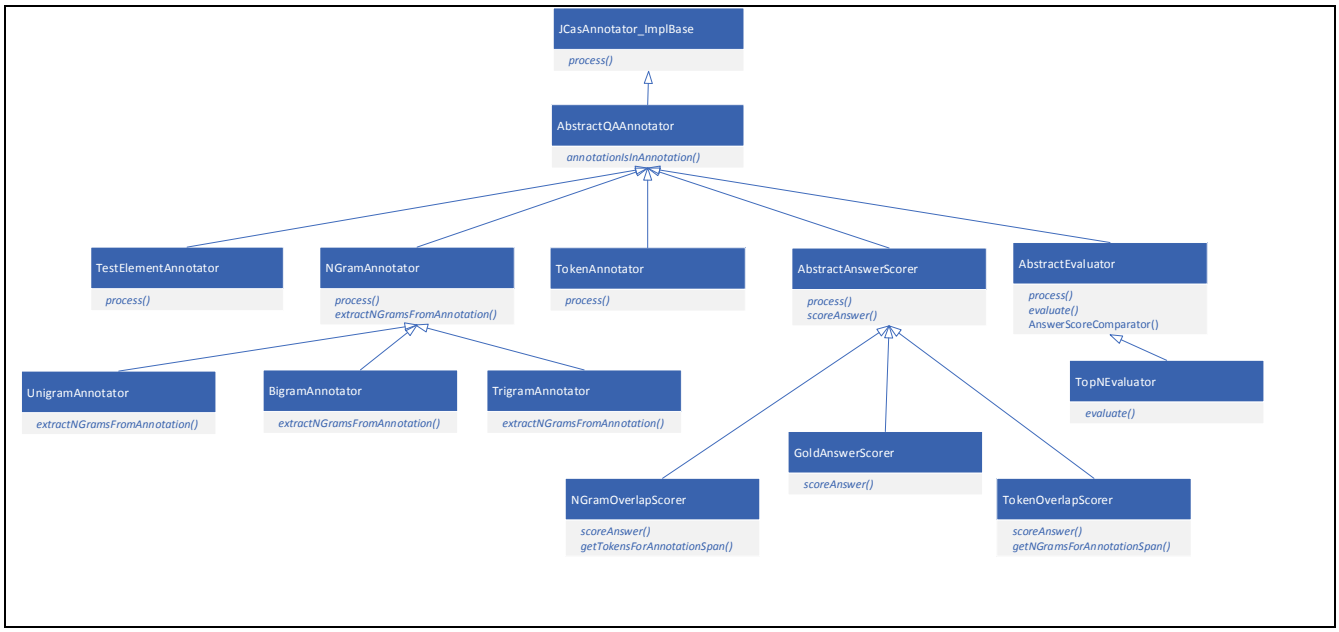
Input Annotations	Output Annotations
AnswerScore	-

The TopNEvaluator extends the AbstractEvaluator and implements the *evaluate()* method. The *evaluate()* method first finds how many correct answers, N, are in the answer set. It then ranks the answers based on their answer scores, and looks at the results of the top N answers. The TopNEvaluator assigns a precision score based on how many of these top N answers are actually correct.

Implemented Methods	Parameters	Comments
evaluate()	**AnnotationIndex** answer scores	Ranks all answer scores and calculates the precision of the top N answers, where N is the expected number of correct answers.

2.5 Inheritance Diagram

The following diagram illustrates how the Annotators described above inherit one another.


JCasAnnotator_ImplBase
process()
AbstractQAAnnotator
annotationIsInAnnotation()
TestElementAnnotator
process()
NGramAnnotator
process()
extractNGramsFromAnnotation()
TokenAnnotator
process()
AbstractAnswerScorer
process()
scoreAnswer()
AbstractEvaluator
process()
evaluate()
AnswerScoreComparator()
UnigramAnnotator
extractNGramsFromAnnotation()
BigramAnnotator
extractNGramsFromAnnotation()
TrigramAnnotator
extractNGramsFromAnnotation()
TopNEvaluator
evaluate()
GoldAnswerScorer
scoreAnswer()
NGramOverlapScorer
scoreAnswer()
getTokensForAnnotationSpan()
TokenOverlapScorer
scoreAnswer()
getNGramsForAnnotationSpan()