11-791 Homework 2 Report

Chen Sun

Andrew ID:chens1

1. System Design Analysis

1.1 Overview

The system controls a simple text document parsing, and uses proper implemented method to parse the answer part of the documents to determine the proper answer to the question given in the documents. Each method should give each answer analyzed a score between 0 and 1 to indicate the relevance of the answer to the question. Finally, an output method should output the sorted method by the score in descending order and calculate the precision at N, where N is number of right answers in the given text.

1.2 Element Parsing and Extraction

1.2.1 Question and Answer

Question and Answer are the most basic elements of the texts. Each question starts with an indicator "Q" and occupies the whole line. Each Answer starts with an "A" and a numer 0 or 1 to indicate whether the answer is a correct answer, followed by a question sentence. There should be a base type named Sentence so that both Answer and Question should implement this. However, since Answer and Question have the shared feature specified in Base_Implementation, thus is no need to determine an extra type to contain these two.

1.2.2 Token

Tokens are elements of Sentences (namely the Answers and Questions). In the text analyzer here, tokens in Question(s) and tokens in Answers have truly different roles and should be specified the Answer or Question they belong to.

With the Question and Answers we get in the 1.2.1, we can use them as input and do the tokenization on Question and Answer fields. Also, to give each question and answer the correct hint the tokens they possess, I introduce two intermediate data types here, the QuestionTokens and AnswerTokens here. For one document, it may have one question and multiple answers, thus there are one Question type and several Answer types. As a result, it may produce one QuestionTokens type showing the question tokens, and multiple AnswerTokens. To distinguish the Answer each token belongs to, a Answer field is needed and added in the AnswerTokens.

1.3 Scoring Methods

With the basic elements extraction finished, based on the extracted tokens and QuestionTokens / AnswerTokens to do the document analysis.

1.3.1 Golden Pipeline Scoring

Golden pipeline method based on the Answer's boolean indicator, which should be regarded as the most ideal situation for the document analyzer. Basically, use the Answer as the input, and read its isCorrect area to get whether it is right. It gives each right answer a score of 1 and each wrong answer a score 0, so that it can distinguished between the right and wrong.

1.3.2 Token Overlap Scoring

Token Overlap based on the tokens in each Answer, it counts the tokens that overlaps with the ones in the Question, and calculates the score by #overlapped/#total answer tokens to get score between 0 and 1. This method, however, doesn't care about the position of tokens in the sentence.

1.3.3 N-Gram Overlap Scoring

The N-Gram uses the neighbored tokens to generate phrases and counts the overlapped N-grams. In this analyzer, 1-Gram (this could also be regarded as token overlap), 2-Gram and 3-Gram are used and counted. For example, a four-word sentences has 9 N-Grams (4 1-Gram and 3 2-gram and 2 3-gram). So basically, N-grams could be regarded as the combination of tokens. Here an N-Gram type is needed to show the range of the token combination and range of N-grams. Also, the number of elements in N-Gram element array could also represent the tokens it has, namely the “N” of N-Gram

1.3.4 Input and Output

The three methods are based on Answers and tokens. Since I've introduced the AnswerTokens which have both answer tokens and an answer instace, thus AnswerTokens could be used to analyze the tokens. Also, the latter two methods need the comparison with the tokens of Question, thus the QuestionTokens could also bed used here.

As for output, each answer should have a score associated with it no matter what scoring method is used here, thus AnswerScore type is used in the process of each comparing method. At last, all the AnswerScore instances are added into JCas index for final output.

1.4 Final Output

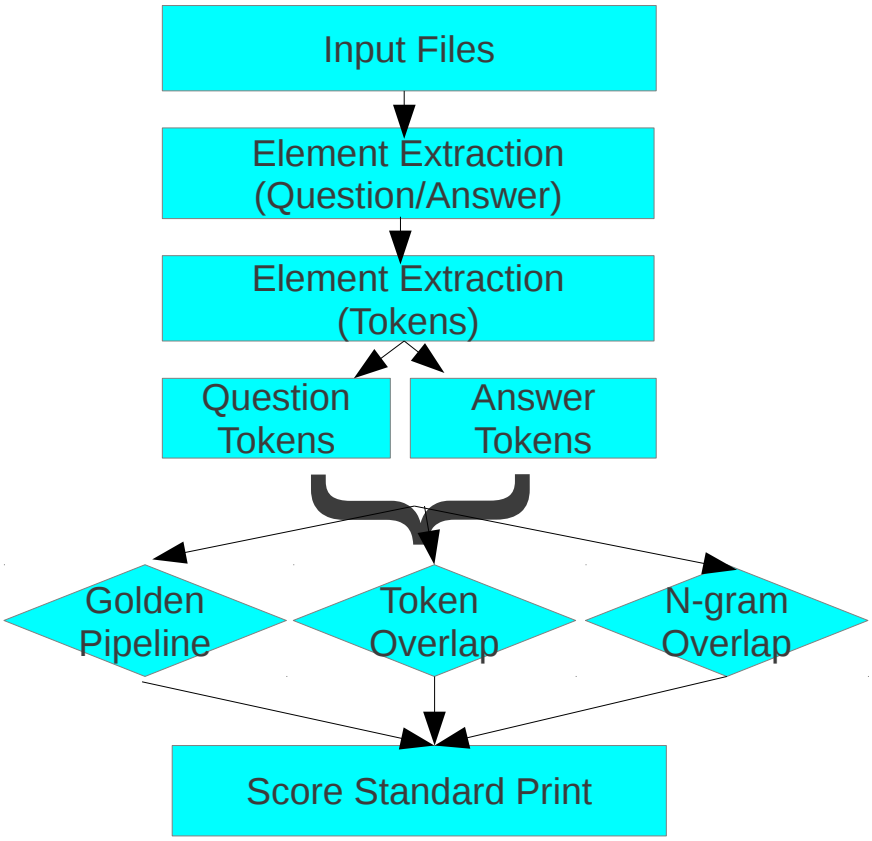
With whichever method implemented the comparing part, the final output needs a standard and universal output. Thus, an output annotator is in charge of this. It reads all the AnswerScore in the JCas document index, sort the answers by the score and output with the question, score and whether it is right.

Also, it counts the right number of answers in each document and calculates precision at N. Finally, when all the documents are processed, it prints out the general average precision of all documents when

the processing is over.

2 Data Flow of Analysis Engine

The gram below shows the data flow of the Analysis Engine.



From the gram above, one thing worthing noting is that users who run the analysis engine has to manually set one method descriptor from the three (which have been provided in the analysis engine) to run one scoring method. There is no support for more than one descriptor at one time.

Also, although the data flow dives into two part after the basic element token extraction, the QuestionTokens and AnswerTokens are actually extracted in one class with one descriptors to control, which mainly because the following steps need both QuestionTokens and AnswerTokens to do the processing, thus there is no need to separate them.

There is also an alternative way omitted by the analysis engine. The user could directly jumps to the Golden Pipeline after the extraction of Answers and Question. This also directly ignores the analyzing stage of tokens.

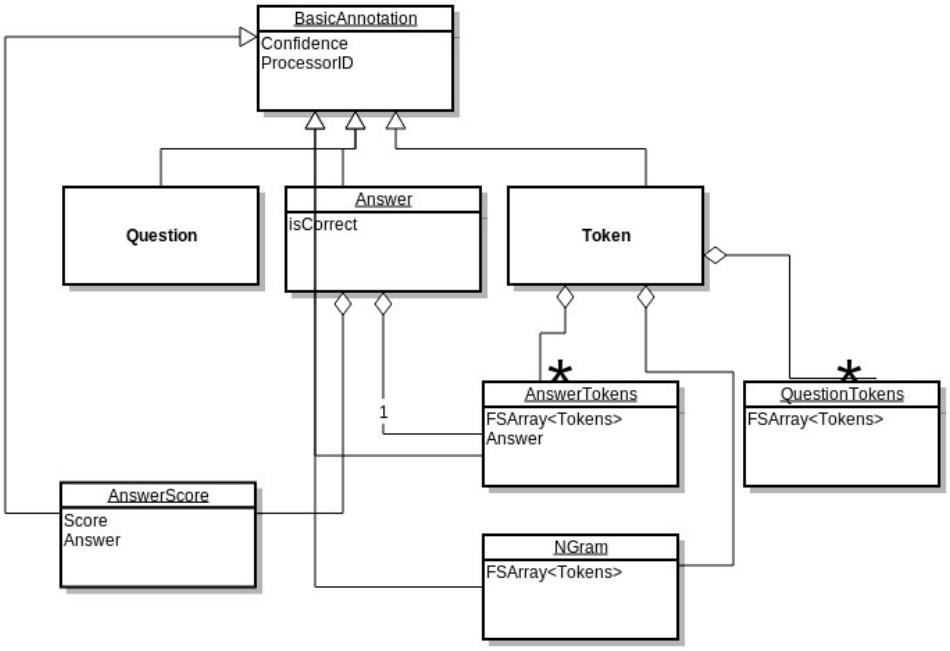
3 Implentations

3.1 Data Types

Data types are presented by TA in the homework specification. To make the job easier, as I have mentioned above, I introduce two intermediate data types QuestionTokens and AnswerTokens, which basically contain an array of tokens belonging to the same Answer or Question. In Answer Tokens, the answer those tokens belong to is also included as a field so that the following processing steps won't mix up the tokens to do whether the scoring or comparing.

Both AnswerTokens and QuestionTokens extends the basic Annotation type defined in the project.

The type system shows as follows.



3.2 Annotators

3.2.1 TextElementAnnotator

The TextElementAnnotator is the initial step of text processing. It reads in the whole text as a whole string. Then it splits the string with “\n” to get the data of each line. For a Question, it always starts with “Q” and an answer always starts with a “A 1” or “A 0”. With this stable format, I could easily extract the question or answer text. For an answer, the prefix 0 or 1 indicates whether the answer is correct, thus reads the token and set the isCorrect field of the answer.

3.2.1 TokenAnnotator

The token annotator reads in the Question and Answer input and for each Question and Answer, parse the covered text area to get the tokens. Meanwhile, it examines the sources of the tokens, to judge the tokens arrays it should be into. Basically, this step finishes the tokenization and output the tokens. Also, the index now has the QuestionTokens and AnswerTokens instance so that it is ready for the next step.

3.2.3 GoldPipelineAnnotator

The GoldPipelineAnnotator reads the answers in the index, and reads the isCorrect area to know whether the answer is right or not. Then each answer is wrapped in an AnswerScore instance.

3.3 TokenOverlapAnnotator

It reads the QuestionTokens and AnswerTokens. For each AnswerTokens, it iterators all the the tokens in an AnswerTokens, and compare each token with the tokens in the Questions. Count all the appeared tokens and calculates the scores with all the tokens in the AnswerTokens. A new AnswerScore is generated containing the answer and the score into the index.

3.4 N-gram Annotator

Actually N-gram annotator contains two parts: NgramAnnotator, which is charget the N-Gram generation and N-Gram Scoring. A An NgramAnnotator iterates the QuestonTokens and AnswerTokens to generate all the 1-Gram, 2-Gram and 3-Grams and add them into index. An N-Gram scoring reads the N-Grams and finds which answer or question they belong too. Then use the comparing method specified in the slides, it then calculates the scores.

3.5 ScorePrintAnnotator

Score print annotator reads in the AnswerScores, whichever annotator generates them. Then it generates a new list to store the sorted AnswerScores by the score each answer has. Finally, it uses the system default print method to print out the result including sorted answers and precision at N.

When it reads all the documents, namely when it is time to destroy, it also prints out the average precision calculated over each document's precision.

4 Run Configuration

Each annotator above has its own descriptor. And the N-gram has two descriptor in an aggregate analysis engine descriptor.

To run different methods, the user needs to manually change the **hw2-chens1-aae.xml** in the file with specified.

The table below shows the mapping of annotator with descriptors.

Annotator(edu.cmu.hw2.annotator)	Descriptor(src/main/resources/descriptors)
TextElementAnnotator	hw2-chens1-text.xml
TokenAnnotator	hw2-chens1-token.xml
TokenOverlapAnnotator	hw2-chens1-tokenOverlap.xml
GoldPipelineAnnotator	hw2-chens1-goldPipepine.xml

NGramAnnotator	hw2-chens1-ngram.xml
NGramScoring	hw2-chens1-ngramScoring.xml
ScorePrinter	hw2-chens1-scorePrinter.xml

Finally, hwe-chens1-ngram-aae aggregates the hw2-chens1-ngram.xml and hw2-chens1-ngramScoreing.xml to finish one single operation.

And the hw2-chens1-aae.xml is the aggregate engines to execute all the flows.

Please add or delete the method with the corresponding method you want to use and make sure scorePrinter is at the last step.

5 Problem

This homework also has very unclear document and requirement of implementation like homework 1. I have to check the the documents over and over again. Also, limited by my knowledge, I don't have much new method to implement.

6 Reference

1. Homework 2 Logical Architecture and UIMA Analysis Engines Design & Implementation
2. UIMA Tutorial and Developers' Guides
3. 11-791 Session 2: Intelligent Information Systems