HW2 - Report

Gabriela N. Góngora S.

1. Task to perform

Implement the logical architecture for the sample information processing task from previous homework. Based on a standard type system given in the new archetype, the task is to create analysis engines and annotators to annotate the sample input files.

2. Processing Pipeline (given):

The information processing pipeline will consist of the following steps:

1. *Test Element Annotation:* The system will read in the input file as a UIMA CAS and annotate the question and answer spans. Each answer annotation will also record whether or not the answer is correct.

2. *Token Annotation:* The system will annotate each token span in each question and answer (break on whitespace and punctuation).

3. *NGram Annotation:* The system will annotate 1-, 2- and 3-grams of consecutive tokens.

4. *Answer Scoring:* The system will incorporate a component that will assign an answer score annotation to each answer. The answer score annotation will record the score assigned to the answer.

5. *Evaluation:* The system will sort the answers according to their scores, and calculate precision at N (where N is the total number of correct answers).

3. Design

3.1. TestElementAnnotator

Input text file comes in, and this annotator has to detect whether each line corresponds to a Question or an Answer. If it starts with a "Q" it belongs to a questions, otherwise, if it begins with an "A" it belongs to an answer.

Input: input file (as UIMA CAS).
Output: Question and Answer (annotations).

3.2. TokenAnnotator

In this annotator we have to go through every token in each question and answer, in order to remove or split from the text any whitespace, punctuation marks, commas, colons, quotes, etc. These characters are to be removed through regular expressions.

Input: Question or Answer.
Output: Token.

3.3. NGramAnnotator

We acquire the tokens generated in the previous stage of the pipeline in order to construct lists of characters, constituting: 1-gram, 2-grams and 3-grams.

Input: Token.
Output: NGram.

3.4. AnswerScoringAnnotator

For this part we could use various similarity indexes such as Jaccard, Tanimoto, Cosine, etc. We are going to use Jaccard's index, as it is a very simple one to implement, as well as very used index for this purpose, plus we have some experience using it. In order to implement this similarity index, we need to have the set of Ngrams corresponding to the Answers and the set of Ngrams corresponding to the Question, therefore we can calculate the similarity between these sets.

$$J(A,B) = \frac{|A \cap B|}{|A \cup B|}$$

Jaccard's index consists in having two sets (e.g. ngrams corresponding to one question and one answer) and dividing the intersections of this two sets by their union, therefore obtaining a similarity measure between sets.

Input: NGram and Answer.
Output: AnswerScore.

3.5. EvaluationAnnotator

This is the last annotator in our pipeline. We need to use the AnswerScore obtained in the previous stage of the pipeline in order to sort Answers by this score. Afterwards we need the top N answers, and we then calculate the amount (%) of answers that are actually correct. The final outcome of the pipeline involves a printing of this percentage as the precision@N.

Input (types): Answer and AnswerScore.
Output (types): precision@N printed on screen.

3.6 Features to be added

To make the implementation easier, the following Features will be added to the type system given:

Type: Token
Feature added: token_type
Description: Identifier for the token. If it comes from an Answer it will contain an "A", else it will contain a "Q".

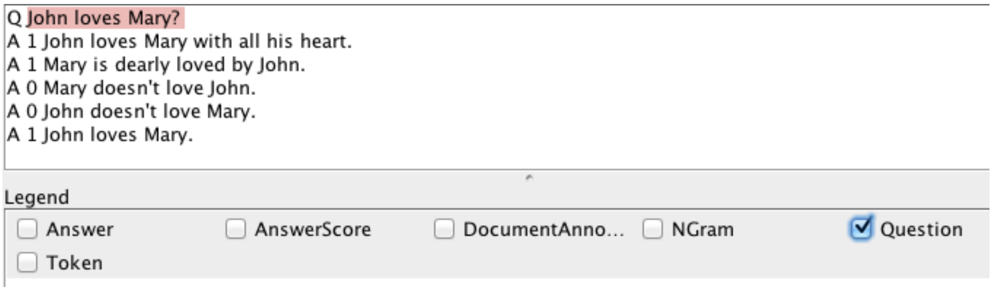
Type: Token
Feature added: line_doc
Description: Identifier for the token, to which line in the document the token corresponds to.

Type: NGram
Feature added: line_doc
Description: Line in the document to which the NGram corresponds to.

4. Results

Some results are shown, to demonstrate the implementation taken place.

TestElementAnnotator

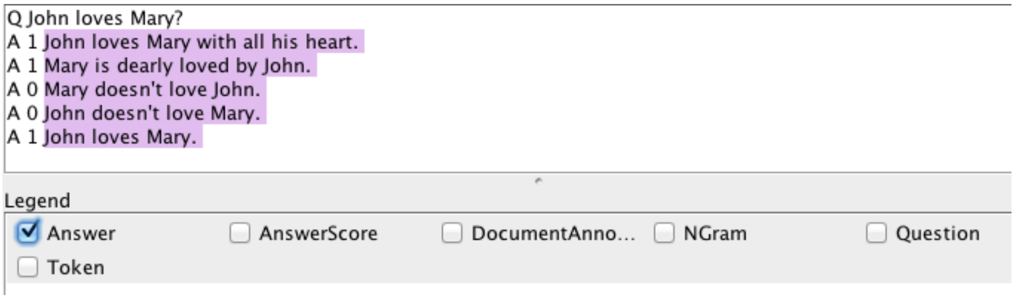


Q John loves Mary?
A 1 John loves Mary with all his heart.
A 1 Mary is dearly loved by John.
A 0 Mary doesn't love John.
A 0 John doesn't love Mary.
A 1 John loves Mary.

Legend

□ Answer □ AnswerScore □ DocumentAnno... □ NGram ☑ Question
□ Token

Figure 1: Question Annotations



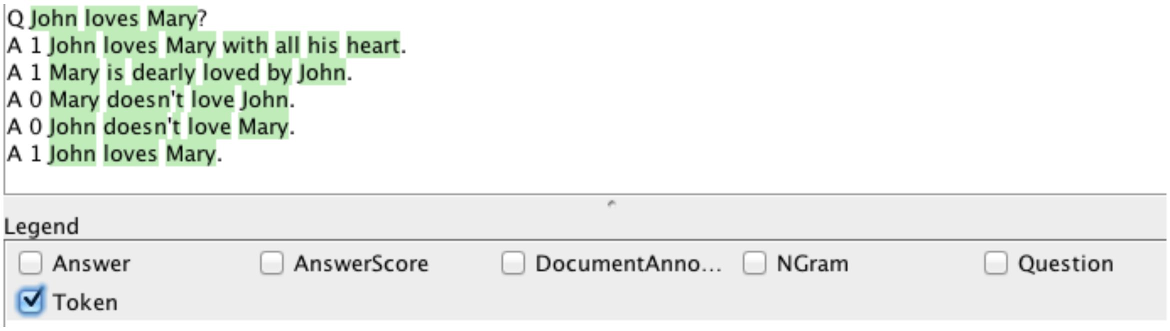
Q John loves Mary?
A 1 John loves Mary with all his heart.
A 1 Mary is dearly loved by John.
A 0 Mary doesn't love John.
A 0 John doesn't love Mary.
A 1 John loves Mary.

Legend

☑ Answer □ AnswerScore □ DocumentAnno... □ NGram □ Question
□ Token

Figure 2: Answer Annotations

TokenAnnotator



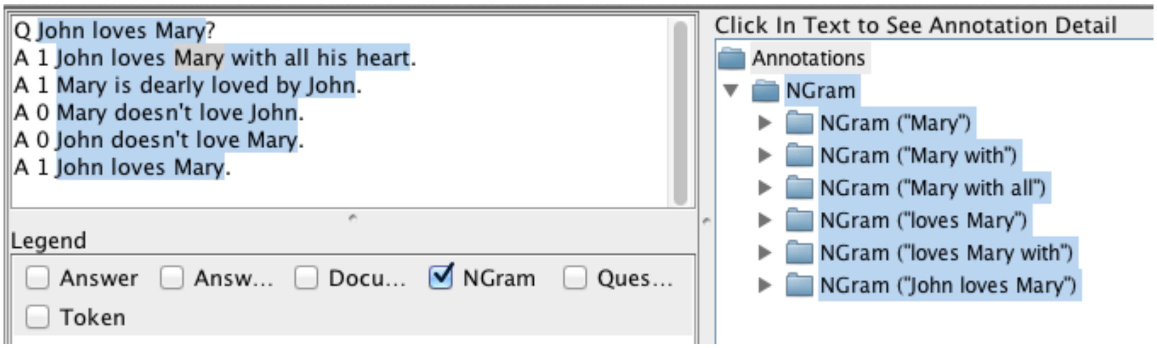
Q John loves Mary?
A 1 John loves Mary with all his heart.
A 1 Mary is dearly loved by John.
A 0 Mary doesn't love John.
A 0 John doesn't love Mary.
A 1 John loves Mary.

Legend

□ Answer □ AnswerScore □ DocumentAnno... □ NGram □ Question
☑ Token

Figure 3: Token Annotations

NGramAnnotator



```
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.
```

Legend

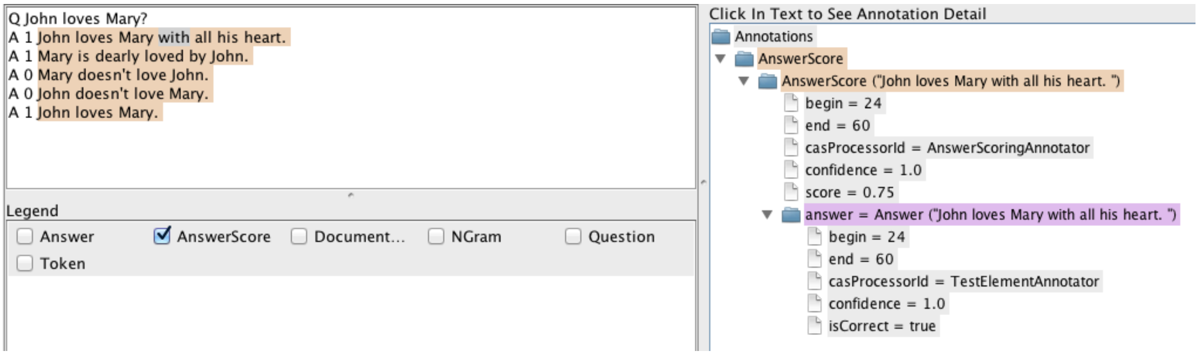
☐ Answer ☐ Answ... ☐ Docu... ☑ NGram ☐ Ques... ☐ Token

Click In Text to See Annotation Detail

- Annotations
- NGram
  - NGram ("Mary")
  - NGram ("Mary with")
  - NGram ("Mary with all")
  - NGram ("loves Mary")
  - NGram ("loves Mary with")
  - NGram ("John loves Mary")

Figure 4: NGram Annotations

AnswerScoringAnnotator



```
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.
```

Legend

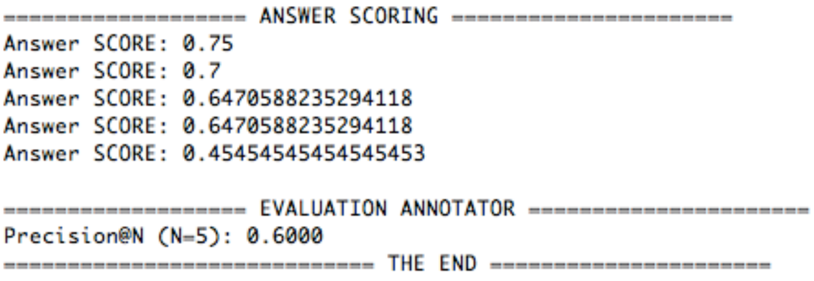
☐ Answer ☑ AnswerScore ☐ Document... ☐ NGram ☐ Question ☐ Token

Click In Text to See Annotation Detail

- Annotations
- AnswerScore
  - AnswerScore ("John loves Mary with all his heart. ")
    - begin = 24
    - end = 60
    - casProcessorId = AnswerScoringAnnotator
    - confidence = 1.0
    - score = 0.75
    - answer = Answer ("John loves Mary with all his heart. ")
      - begin = 24
      - end = 60
      - casProcessorId = TestElementAnnotator
      - confidence = 1.0
      - isCorrect = true

Figure 5: Answer Scoring Annotations

EvaluationAnnotator



```
=================== ANSWER SCORING ======================
Answer SCORE: 0.75
Answer SCORE: 0.7
Answer SCORE: 0.6470588235294118
Answer SCORE: 0.6470588235294118
Answer SCORE: 0.45454545454545453

=================== EVALUATION ANNOTATOR ======================
Precision@N (N=5): 0.6000
============================= THE END ======================
```

Figure 6: Final evaluation

5. Discussion

After implemented the design described above the following considerations might want to be taken for future work on this project:

1) The apostrophe represents quite a challenge for this task. For example the word "doesn't" actually means "does not", but in order to perform this more rigorous distinction we would have to implement other NLP techniques, such as lemmatization and stemming. The current implementation replaces the apostrophe for a whitespace and two tokens are made "does" and "t".

2) Perhaps for a more versatile implementation the “N” in the “NGramAnnotator” should be set as a parameter and that way various values of N could be tested, not just 1,2 and 3-grams.
3) The way in which answers scores are obtained could use some refinement, future weight could involve the configuration of various scoring techniques, not just Jaccard’s index, and even a more extended implementation of Jaccard, for example a weighted version depending on the value of N in the Ngrams could give better results.