11-791 HW2: Logical Architecture and UIMA Analysis Engines Design & Implementation

Xiaohua Yan

Language Technologies Institute

In this homework we are required to implement the analysis engines for the information processing pipeline described in Homework 1 based on the type system we designed for the task. In the following sections I explain in more details about the implementation of each analysis engine component of the pipeline.

Test Element Annotator

In the first step of the pipeline, each question (starts with "Q" in the document) and answer (starts with "A" in the document) are annotated based on their start and end indexes. Distinguishing between questions and answers can be simply by looking at the first symbol of each line of input document. However, we should not assume that the text string of each question and answer starts at fixed index since it is possible that there could be multiple delimiters (e.g. whitespace, tab, etc.) between the identifier (i.e. "Q" or "A") and the actual text content. Instead, I did some preprocessing for the string after the identifier by omitting the leading and trailing whitespaces using the `String.trim()` function to ensure finding the correct start and end indexes of a sentence.

Token and Sentence Annotator

In the implementation of the Token and Sentence Annotator, I made use of the annotation result of Test Element Annotator because the annotation of questions are answers are helpful for annotating the generic Sentence type, which is further used to annotate the tokens in each sentence.

To be specific, I first obtain the start and end indexes of each sentence based on the annotation results of Test Element Annotator. Then tokenization is performed on each sentence using the regular expression I designed, which can be described as

```
Pattern tokenPattern = Pattern.compile("[\\w\'-]+|$*\\d+.\\d+");
```

The regular expression is able to tokenize English words, punctuations and digits which produces the correct tokenization for the example input documents. Admittedly, the regular expression is far from robust. Due to the limited time for the assignment, I did not think of a more careful designed regular expression. An alternative may be to deploy the tokenizers in off-the-shelf libraries such as the Stanford Tokenizer (http://nlp.stanford.edu/software/tokenizer.shtml). However, I chose not to use the libraries with better tokenizers simply because I wanted to get experience in developing the whole processing pipeline.

N-gram Annotator

The N-gram annotator makes use of the annotation results produced by the Token annotator. One interesting thing for this annotator is that I need to set the order of the n-gram as a parameter, which draws value in {1, 2, 3}. By iterating over the tokens in each sentence, I am able to get the n consecutive tokens in a row, which is further stored in FS Arrays, each element of which is a single token.

Answer Score Annotator

The Answer Score Annotator computes two kinds of scores when comparing the question with each question. The first score measures the token overlap based on cosine similarity between the question-answer pair represented by two bags of words. For example, assume the question is

```
Booth shot Lincoln?
```

And the answer is

```
Booth was shot by Lincoln
```

Then the two sentences can be represented as where the numbers indicate

	Booth	shot	Lincoln	by	was
Question	1	1	1	0	0
Answer	1	1	1	1	1

the occurrence frequency of each token. And the cosine similarity of two vectors is defined as

$$\text{similarity} = \cos(\theta) = \frac{A \cdot B}{||A||||B||} = \frac{\sum_{i=1}^{n} A_i \times B_i}{\sqrt{\sum_{i=1}^{n}(A_i^2)} \times \sqrt{\sum_{i=1}^{n}(B_i^2)}}$$

where A_i and B_i indicate the elements of two vectors A and B.

The other score measures the n-gram overlap score computed by the arithmetic average of the cosine similarity between all n-gram pairs in each question and answer.

And finally the two scores are generated based on a simple linear combination of the two scores, the weights being 0.8 for token overlap and 0.2 for n-gram overlap respectively.

Precision Evaluator

After all the annotations are finished, I was able to evaluate the performance of the scoring method I implemented in the Answer Score Annotator since the ground truth of the correctness of each answer is already given. And the final precision at N, where N is total number of correct answers, is printed in console for reference.