11791 Homework 2 report - Analysis engine design and implementation

Zhengzhong Liu
Language Technology Institute
Carnegie Mellon University
Pittsburgh, PA 15213

In this homework, we are required to design and implement a sample Question Answering task. The analysis engine design this time learn some basic work flow from homework1's suggestion, which is listed as followed:

1. Test Element Annotator
   A test element refers to either a question or an answer. The test element annotation processing the document and identify which span of the text compose the question and which part construct an answer. The given input is assumed to have only one question and multiple candidate answers for this question, following the specific formatting (i.e. the single letter indicator at the start of one line saying ”Q” or ”A”). From the requirements, we see that most of the required information are simply contained in the source document, which implies that no prerequisites should be needed.

   Another label in the input document is the correctness of each answer, where 0 indicates wrong and 1 indicates correct. This annotator also capture this information and store label the Answer annotation with its correctness information by using the ”isCorrect” feature.
2. Basic language unit Annotator
   There are several basic language units that are often used in analysis in both computer science and linguistic terms, such as Tokens (or words), Sentences. Token will also be a prerequisite for many follow up annotators, including NGram and evaluation annotators. Thus this design put such annotators at the early stage of the pipeline. The Stanford CoreNLP toolkit is used in this task, the annotators adopted from the pipeline includes: tokenize, ssplit, pos, lemma, ner, which perform tokenization, sentence splitting, part of speech tagging, lemma identification and named entity recognition. The offsets from the output of the Stanford pipeline are then calculated and mapped back the the JCas document text. Tokens, entity mentions and sentences are annotated in the JCas.
3. NGram Annotator
   A NGram annotator essentially combines consecutive tokens as one single element. The annotator design considered the different possibilities of N. To make the annotator flexible, the user is able to provide an multi-value parameters, indicating which ”N”-gram the system should annotate. From a linguistic perspective, it does not make sense for NGram to cross sentence boundary. Thus the annotator will process each sentence a time, the algorithm in the annotator will ignore the N which is longer than a sentence.
4. Answer Scoring
   Answer scoring is a complex task, however, in this homework, only limited information is given. There are only limited examples for questions, it is unclear what type of questions will be contained in the final task, and what information will be the most important in scoring. Here the system adopt a minimum method which only uses unigram similarities between the question string and the answer string, as more complicated method does not garauntee to produce better performance. It is also difficult to decide at implementation time, how to combine score from different sources appropriately (for example, how important should a unigram be, and how important should a bigram be.)

The overlapping method on class concerns the length of the question and answer, this is not an easy definition because it is unclear what implies ”long question”, thus the system adopt the cosine similarity measure of the bag of words of the question and the answer. The bag of words is constructed unigrams, and the term frequency are computed locally in each question or answer. Inverse document similarity is not used here. Cosine similarity has the ability to normalize the score by length, thus naturally embed the consideration of length into score.

The final cosine similarities between a question and an answer is considered as the confidence of this answer and will be put into the confidence feature of the answer. The casProcessorId feature is changed to the complete class name (which will potentially help future development to choose from one answer the most appropriate annotators).

5. Evaluation
   There are no evaluation data type in the given type system. Here the design also did not store the evaluation result into the system. However, one evaluation annotator will be used to gather the answers provided by the previous annotators, and sort them in terms of their confidence. Precision at N is used to perform the evaluation.

To summarize, the whole aggregate engine combines the following annotators:

1. TestElementAnnotator : annotate the question and answer types
2. StanfordCorenlpAnnotator : annotate the tokens, sentences, named entities
3. NGramAnnotato : annotate NGram based on user provided parameters
4. CosineBasedAnswerRanker : annotate answer confidence using cosine similarity
5. PrecisionAtNEvaluator : a simple evaluation that calculates precision at N based on the confidence

The main design principles are lied in annotation types. Here one principle for this is to use one annotator to annotate only related types, in best case only one type. This design principle ensure flexible to replace annotators, and easy to identify problems. The casProcessorId type help the system to know which annotator annotate the document, thus provide information for future fusion. One exception here is the Standford Corenlp based annotator, which annotate both token, sentence and entity mentions, this is because the overhead for initialize stanfordCorenlp is non-trivial, thus initialize it multiple types is not an efficient solution.