Homework 2

Salvador Medina
ITAM
Design of Intelligent Information Systems
2014-03-07

Requirements

The purpose of this homework was to create an aggregated analysis engine (AAE). The AAE main task is to analyze through n-grams to detect from a set of answers given to a question which ones are correct. The pipeline required for the AAE is as follows:

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).

Type System

Given the set of requirements, it was decided to modify the type system described in the base project in order to accomplish the annotations and results desired.

First of all, the *NGram* type was modified to have the *elements* and *n* attributes. The *elements* attribute stores an array of n-grams, while *n* indicates the size of the n-gram window.

Afterwards, the *QuestionTokens* and the *AnswerTokens* type was added. This type stores the corresponding Question and Answer annotation while also storing in an FSArray all the tokens for each Question or Answer.

In a similar manner the *QuestionNGram* and *AnswerNGram* type were declared. However, in this type instead of storing an FSArray of tokens, it stores the unigram, bigram and trigram of the corresponding Question or Answer.

Aggregated Analysis Engine

Based on the requirements, an AAE was designed on 5 annotators. The annotators were created as primitive analysis engines. Each annotator is briefly described as follows:

1. ***Input Annotator***: Reads the input file and annotates the declared question and answers.
2. ***Token Annotator***: Tokenizes every Question and Answer annotation, considering blank-spaces and punctuation marks as the delimiters of the tokens.
3. ***N-Gram Annotator***: Builds the unigrams, bigrams and trigrams of every Question and Answer annotation, based on their tokens.
4. ***Scoring Annotator**:* Sets a score to each answer according to the n-gram analysis. The n-gram comparison was done by measuring the Jaccard similarity of the n-grams.
5. ***Evaluation Annotator**:* Calculates the precision with different number of N correct results.

The *Input Annotator* reads the input file, parses the file according to the specification using regular expressions and annotates the text corresponding to the question and the answers. For the answers, during the parsing it is also detected whether the answer was classified as correct or incorrect through a 1 or 0, respectively. This value is stored within the Answer annotation. Therefore, Question and Answer annotations are stored in the index up to this point.

In the *TokenAnnotator* the Question and Answer annotations are retrieved from the index. The question and answers found are then tokenized by splitting the strings. The space, comma, period and semicolon are used as delimiters to tokenize the strings. The tokens are annotated within a Token type. These Token annotations are grouped within an FSArray, which is part of the QuestionToken and AnswerToken annotations. Once these annotations are built, they are added to the general index of the AAE.

For the *NGramAnnotator*, the *QuestionToken* and *AnswerToken* annotations are retrieved from the index. For each annotation, its corresponding unigram, bigram and trigram is built. The n-grams are built based on words, given that the tokens are defined as the words of each sentence. The three new n-grams are set to a new *QuestionNGram* or *AnswerNGram* annotation, depending on each case. Finally these annotations are stored in the index.

The *QuestionNGram* and *AnswerNGram* are consumed by the *ScoringAnnotator*. In this annotator, the n-grams ($n = 1,2,3$) of the question are compared with each answer. The similarity measure used within this project was Jaccard's similarity. In order to measure the similarity, the repeated n-grams were removed. Considering that the S_1 corresponds to the set of the question n-grams, while S_2 is the set of the answer n-grams. Therefore, the score was calculated as follows:

$$score = \frac{S_1 \cap S_2}{S_1 \cup S_2}$$

The QuestionNGram was reduced to a simple Question annotation, while the Answer results were indexed within an AnswerScore annotation.

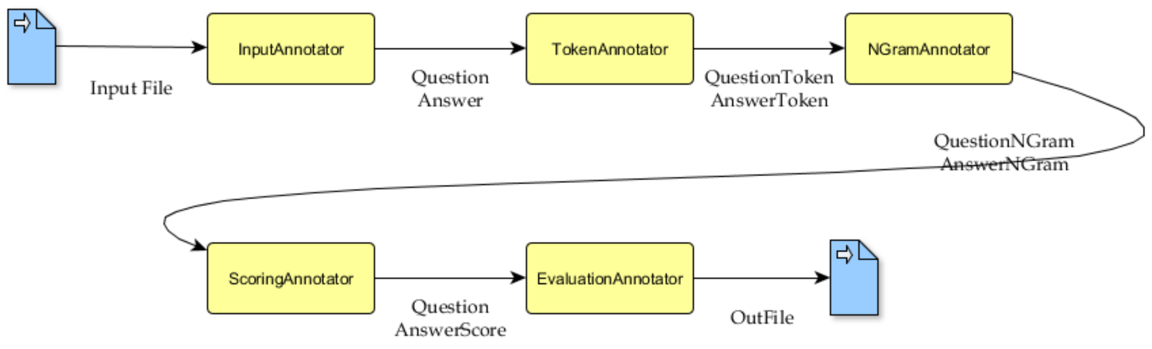

Figure 1 – Aggregated analysis engine pipeline

Finally the AnswerScore annotations are obtained from the index. The answers are decreasingly ordered within an array list. Finally, the score is translated into a binary result (correct or incorrect) based on a threshold set to 0.5 and compared to the original correctness of the answer. Finally the results are printed to the system output and the file Results.txt.

Results

The AAE developed was executed three times. Each time for a different n-gram which gave the following results while evaluating the *q001.txt* file.

Table 1 –Results

	Precision		
N	***1-gram***	***2-gram***	***3-gram***
1	1.00	1.00	1.00
2	0.67	1.00	1.00
3	0.57	1.00	1.00
4	0.50	0.67	0.67
5	0.00	0.71	0.71

Even if the results seem favorable, the truth is that the recall should be really low. The reason why at N=1 the precision is 100% accurate is because there exists an answer which is exactly the same as the question: "*Boot shot Lincoln*". However, as the number of words increases and the complexity of the semantics becomes harder to understand, this kind of method is less favorable.

Discussion

An AAE was built successfully meeting the requirements established, with a flexible design which will allow some of the annotators developed can be reused in future projects. Nevertheless, this implementation can be improved. A quick improvement could be done by using a different similarity measure, such as cosine distance. It could also be interesting to tokenize the questions and answers by letters or syllables and compare the results obtained. However, the n-gram method has shown that conceptual similarities cannot be achieved and another method is desirable.