11-791 Project Individual 2: UIMA Type System
Constantine Nakos
September 14, 2015

Overview

The data types specified in this project should be sufficient to serve the needs of the information processing task given to us. The sample task takes test elements as input, each containing a question and a set of correct or incorrect candidate answers. These are then tokenized, grouped into n-grams, and used to assign a score to each of the candidate answers. The scores are then used to rank each of the answers, and the precision of the top N elements of the ranked list is used to evaluate the system's performance.

The data types outlined below suffice to identify the questions and answers in the program's input, annotate the tokens and n-grams within them, and assign a score to each answer. As pointed out in class, the Evaluation step measures system performance and does not rely strongly on a specific data type for its output. In principle, one could design a PrecisionAtN data type to store performance results, but for the sake of this project it would serve little purpose. Thus the last data type that appears in the pipeline is AnswerScore, the output of Answer Scoring and the input of Evaluation.

Types

The types defined for this project are as follows. Features containing redundant information are noted in the text; they are generally included for maximum flexibility in data processing.

EnhancedAnnotation

The EnhancedAnnotation type serves as a base type for the annotations generated in this project. Aside from the *begin* and *end* features inherited from Annotation, the EnhancedAnnotation type has a *componentName* feature and a *confidenceScore* feature. The former is a String containing the name of the component that generated the annotation. The latter is a Double containing the component's confidence in its annotation. All other types in this project inherit from EnhancedAnnotation directly or indirectly.

Sentence

The Sentence type inherits from EnhancedAnnotation and represents a span of text containing a complete thought. Its *begin* and *end* fields should contain offsets within the document to the beginning and ending of the text itself, without any formatting elements. Though not strictly necessary and never instantiated directly, the Sentence type provides a common superclass for Question and Answer that allows both to pass through the same text-processing components.

Question

The Question type inherits from Sentence and contains information about a single test question. The *begin* and *end* fields mark the beginning and the ending of the question's text. The *questionNumber* field is an integer containing the number of the question, typically the file number in which the question was found. (The question number may or nay not be useful to the tester or the end user, depending on the implementation.) The *candidateAnswers* field contains an FSArray of Answer

objects that form the candidate answers provided for the question. Questions are generated by the Test Element Annotation step alongside their corresponding Answers.

Answer

The Answer type inherits from Sentence and contains information about a single answer to a test question. As with Question, the *begin* and *end* fields mark the beginning and ending of the answer's text. The *answerNumber* field is an integer containing the number of the answer within the set of candidate answers. (This field contains the same information as the answer's index within the Question's *candidateAnswers* field but may be more accessible as a field of its own.) The *correct* field is a boolean that is set to True if the candidate answer is correct and False otherwise. This information is contained in the "1" or "0" before the text of the answer in the input files. Finally, the *parentQuestion* field contains a backpointer to the Question which contains this Answer. Again, this pointer is not strictly necessary but allows for flexibility in processing the data. Answers are generated by the Test Element Annotation step alongside their corresponding Questions.

Token

The Token type corresponds to a single token within a sentence under whatever tokenization scheme is used by the implementation. The inherited *begin* and *end* features mark the offset of the beginning and ending of the token within the document. The *parentSentence* feature contains a pointer to the unique Sentence which contains the token; this information is recoverable and is included for ease of acess only. Tokens are generated by the Token Annotation step of the pipeline.

Ngram

The Ngram type corresponds to a sequence of *N* tokens within a Sentence. Ngram inherits from EnhancedAnnotation, and its *begin* and *end* features should contain the offsets of the beginning and the ending of the n-gram within the text, respectively. The *tokens* feature is an FSArray that contains pointers to the Token objects that comprise the n-gram. The Unigram, Bigram, and Trigram types are subtypes of Ngram that correspond to 1-, 2-, and 3-grams respectively. They exist for the purposes of clarity and convenience of n-gram iteration only; they provide no additional features over the base Ngram class. The type specification does not enforce that an Ngram subtype contains the appropriate number of tokens. This behavior should be enforced by the code. Ngram annotations are generated by the N-gram Annotation step of the pipeline.

AnswerScore

The AnswerScore type is an EnhancedAnnotation that represents a score assigned to an Answer by the Answer Scoring step of the pipeline. The *begin* and *end* features should correspond to those of the Answer in question. The *score* feature should contain a real-valued score in the range [0,1]. The *answer* feature should contain a pointer to the Answer the score corresponds to. Note that similar results could be achieved by including a *score* feature within the Answer type, but separating the score out into a separate AnswerScore type allows the assignment of multiple scores to the same Answer using different components. The separation also keeps us from having to modify Answer instances that have already been created.