11-791 Design and Engineering of Intelligent Information System

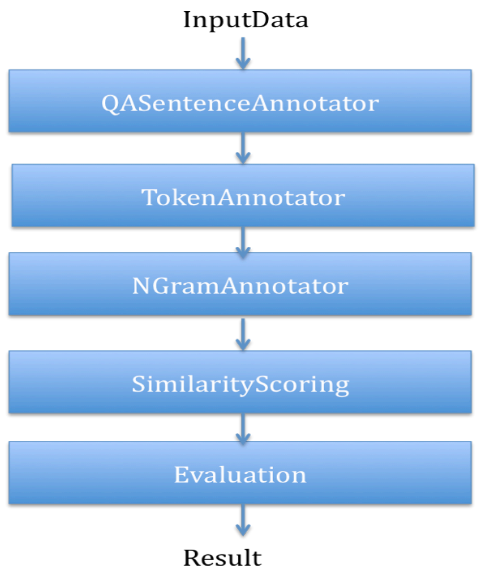
HW #2: Logical Architecture and UIMA Analysis Engines Design & Implementation

Keerthiram Murugesan

Andrew-id: **kmuruges**

Requirement Analysis:

In this homework, we have given a simple Question Answering System task. We have a set of Question-Answers (QA) pairs, where each question contains one or more answers. The maven archetype already specified the default types to be used for the system and we will stick to this notation. The main requirement analysis in this task involves gathering functional requirements for the pipeline. The following figure shows the main components of the system.

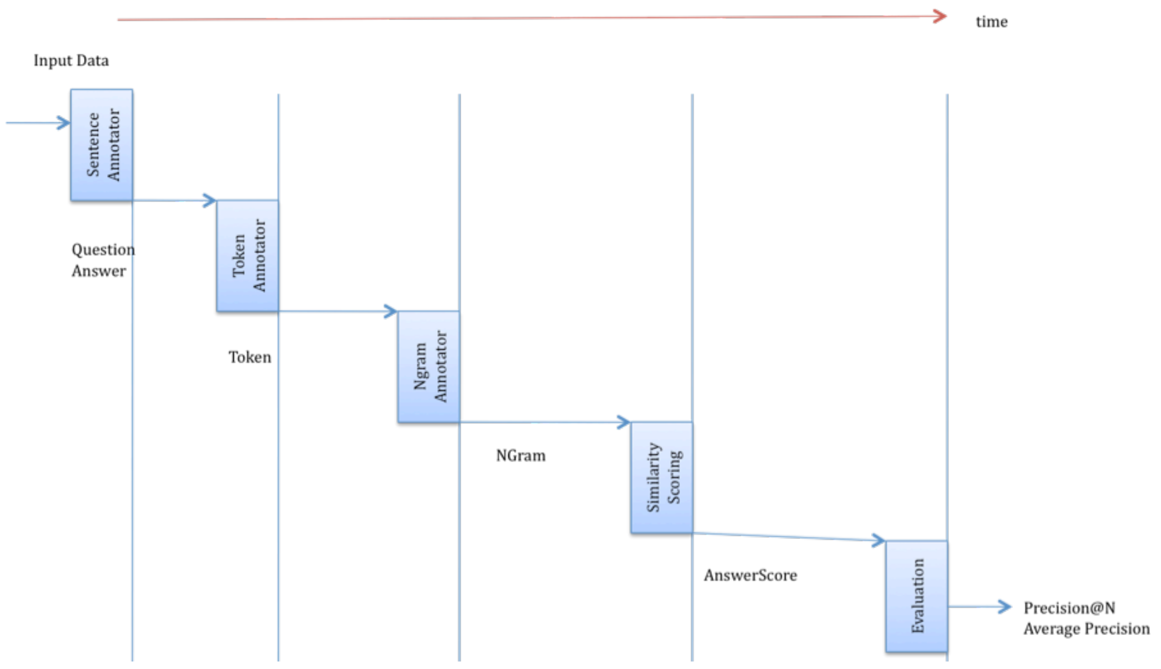


The system should annotate the given QA pair into Question sentence and Answer sentence. These sentences are then tokenized, get corresponding NGrams from these tokens. For simplicity, we restrict ourselves to Unigram, Bigram and Trigram tokens. In addition, we can apply any NLP or machine learning task. These Ngrams are then used to compute the score for each answer. If we have more than one annotation method, we should use a weighted scoring that merge these individual scores to compute final score for the answer. These scores are then used in the evaluation phase to compute precision@N and eventually the average precision value.

Perspective: Annotations Manipulation and QA Evaluation

The entire processing needs are dedicated to creating annotations for the given input document (QA Pair) and use NLP/ML tools to compute how similar an answer is to the given question. The annotations created throughout the pipeline are identified using casProcessorId field in the custom Annotation class. Each Annotation also contains confidence parameter, which is used in the Scoring phase for weighted scoring. More confident the annotation is good for computing similarity, the confidence score will be higher. Figure below shows the Input/Output types used in the system. We use *UIMAFit* to access the annotation types in each annotator of the pipeline.

TestElementAnnotation: It extracts the Question and Answer annotation from the raw input text. The annotation contains the entire sentence including the sentence indicator (Q or A). This is especially useful in the next phase when we need to compute the start and end index for the token annotation. It uses the regular expression to find whether the sentence is of type Question or Answer. The *regex* pattern is passed to the annotator using the UIMA configuration parameter to enable easy configuration in the future.



TokenAnnotator: The Question and Answer inputs from previous annotator are used to generate tokens. We use Stanford CoreNLP library to tokenize/POS tag the sentence. We remove the Q or A tag that is used to identify the types before the annotation (using indices). We especially consider the token that splits words like *doesn't* into *does* + *n't.* These tokens are then added to the annotation index.

NGramAnnotator: This annotator uses the tokens generated from the previous component to generate the unigram, bigram, trigram tokens. It removes any tokens that contains (dot) or (comma) to avoid any unnecessary count to the final score. Each NGram contains a set of tokens based on whether it is unigram, bigram or trigram.

SimilarityScoring: This component uses the NGrams generated in the previous step to compute the similarity between the question ngram tokens and the answer ngram tokens. The same is repeated for POSTokens and LemmaTokens. It uses the NGram Overlap Scoring methods mentioned in the slides.

It follows a simple *bag of word* model. Each operation (such as unigram, trigram, etc.) type is scored separately and the set of these scores are merged using the weighted score function that gives different weight to different operation. The operation weights sum to one. We also use simple heuristics to check the score metric is computed correctly.

Evaluation: This component simply computes the Precision at N from the AnswerScore types generated in the previous step. It sorts the answers based on the computed score and the precision at N metric is computed. We used the *collectionProcessComplete* function to get the average value of the precision value.