Design & Engineering of Intelligent Information Systems: pi2

Instructed by *Dr. Eric Nyberg*

Due on Sep 14, 2015

Vivian Robison vrobison

To implement the required types, I went through the workflow diagram given at the beginning of the assignment writeup and for each annotator, listed what the required input and output would be, and checked that the accumulated metadata at the beginning of a annotator in the given pipeline would be sufficient for that step. I simplified tokens and ngrams into one, as a token is simply a unigram. From this, I got five types to create.

Implemented types:

1. **annotationPlus** The type `annotationPlus` is a subtype of the built-in `Annotation` with the addition of two features: a String feature `componentOfOrigin` that holds the name of the component that created that annotation, plus a Float feature `confidence` that holds the component of origin's confidence in that annotation. The following four types are all subtypes of `annotationPlus`, so they all will have the name of the component that created them and a confidence score. Storing the creating component name as a string allows the type to be reused at different places along the pipeline, which we would not be able to do if the component was part of the type name. Making the following types subtypes of `annotationPlus` avoids having to redefine the confidence score and name string for each type.

2. **question** The type `question` is an annotation that marks a question in the input file. This is output in the test element annotation step.

3. **answer** The `answer` type is an annotation marking the text of a candidate answer in the input. It has a Integer feature `number` to hold the number of the answer, originally given in the form A#. There is also a `truthValue` Boolean feature that is 1 if the answer is correct and 0 if it is incorrect. This is also output in the test element annotation step, and used in tokenization and ngram annotation.

4. **scoredAnswer** The type `scoredAnswer` is a subtype of `answer` that has the additional Float feature `score` containing the score assigned to that answer. This is output in the answer scoring step, and used in candidate ranking.

5. **ngram** I chose to make a type `ngram` as an annotation with an Integer feature `n` specifying if it is a unigram(n=1), bigram(n=2), trigram(n=3), etc. This allows us to expand to larger order n-grams without the creation of additional types, should we later decide they are needed or want to reuse parts of the code for other projects. Alternatively, there could be a type ngram that unigrams, bigrams, trigrams, etc. are subtypes of. I chose to store the n value as a feature for flexibility and reuseability. Tokens are equivalent to unigrams, so there is no need to create a separate "token" type, a token is a `ngram` with n=1. Ngrams with n=1 are output in the tokenization step, and n=2 and n=3 are output in the ngram annotation step. All ngrams annotated are then used to score the candidate answers.