Ryan Carlson (rcarlson)
11791 – HW 1 Report
September 11, 2013

Motivation

In this brief report I describe my Logical Data Model for the given information processing task. To start, let's consider the data format and processing pipeline, which will inform how we structure the type system. Each file will come in the following form:

```
Q <question>
A <isCorrect> <answer>
A <isCorrect> <answer2>
A <isCorrect> <answer3>
...
```

So, at the bare minimum, we need some way to represent questions and answers, and to represent whether or not an answer is correct. We also note that there is a one-to-many mapping between questions and answers. That is, each question has a list of possible answers.

Now that we have some vague idea about how to represent the input data, let's consider the processing pipeline, which has 5 stages:

1. Test Element Annotation
2. Token Annotation
3. NGram Annotation
4. Answer Scoring
5. Evaluation

We've already discussed Test Element Annotation, but now let's consider Token Annotation. Basically, we need some way to represent a string of text as a list of tokens. With NGram Annotation, we'll be extracting NGrams for n=1,2,3, so we need some way of representing that. Also, we'll be scoring the answers, so each answer needs a score. The Evaluation step seems to require a sort step, but that doesn't have any obvious implications in the type system.

So, to summarize, here are our requirements for a type system:

- We need to represent a single question and multiple possible answers
- Every question and every answer is made up of text
- All text should be tokenized
- All tokens should be grouped into unigrams, bigrams, and trigrams
- Every answer should have a score and we should know if the answer is correct (at least at this stage – I assume we won't always know about the correctness of a particular answer)

There is one additional requirement: every component should know its source, and every component should have a confidence associated with it.

Specification

Now that we have the requirements in English, let's define our type system! Let's start with the requirement that every component should have a source and confidence. Instead of annotating every new component with these flags, let's (as the assignment suggests) create a superclass that all new classes will inherit. So our first step is to generate a `SourceConfidenceAnnotation` that inherits `uima.tcas.Annotation`. This new type has two fields: `String source` and `Double confidence`. Assume all new types directly inherit `SourceConfidenceAnnotation` unless otherwise noted.

Now, let's start at the bottom and build our way up. That means starting with the text. We know we need to tokenize and capture NGrams from those tokens, so let's create a new type `TextFeatures` with the following features:

- `String text`
- `StringList allTokens`
- `FSList<NGram> unigrams`
- `FSList<NGram> bigrams`
- `FSList<NGram> trigrams`

The text is just the raw string text passed in from either an answer or a question. The `allTokens` object will separate each token as a `String` in a list. And each of the NGram features is of type `NGram`, which I define as follows: an `NGram` type has integers `startIndex` and `endIndex` (in the original text) and a list of tokens (`StringList`). I am not entirely sure this is the right representation of an NGram, since the tokens alone might be enough, but for now I want the extra context of knowing where in the text the NGram was taken. This may not turn out to be useful (in which case the type system should be changed).

Now that we know how to represent text and textual features, let's think about an answer. For this we create an `Answer` object that, as alluded before, has three features: `Double score`, `Integer isCorrect`, and a `TextFeatures` type that represents the actual text of the answer. Note that `isCorrect` is an `Integer` and not a `Boolean` because I suspect we will not always know whether or not an answer is correct, so I leave a -1 value for unkown (1 means the answer is correct, 0 indicates incorrectness).

Finally, all we need to create is a `Question` object, which just contains a `TextFeatures` object representing the question textual features and a `FSList` of `Answer` objects.

This covers all of the stated requirements from the previous section, so I do believe we have a sufficient type system!