Andrew Wilkinson
DEIIS Fall 2015

"Project Individual" 5 Report: Numbers and such

For this week's assignment, I was able to reuse most of my code from Homework 4. I don't need to repeat everything I wrote a week ago, so I'll just describe what's new.

What this assignment required that the last one didn't was the ability to rank passages based on their score; this is a necessary first step before implementing the code for the required metrics. The problem is that whether one uses FSArray or FSList as the data structure to store passages, along with their attendant features, neither of those data structures as implemented in UIMA is very useful in terms of API or methods. One would like to be able to iterate over them and perform sorts, but that isn't straightforward. So the task was to use a little ingenuity to accomplish that. Even though it's not the most elegant, some kind of workaround was necessary.

In Homework 3, the scores we used to sort the passages were integers: they corresponded to number of matching n-grams between a question and a passage/answer. For that assignment, I was able to perform the ranking just using arrays and for-loops, counting down by 1s from the maximum possible score an answer could have and adding (the string version of) an answer to a sorted array when its score feature matched the value of *i* in the iteration. However, for this assignment, the scores are doubles, which means that method would not work.

What I did was to define a class "Psg" that represented the UIMA "Passage" type defined in the type system, within `InputDocumentAnnotator.java`. I gave this class the ability to hold values in its variables that correspond to the same primitives defined as features for the "Passage" type. I also, crucially, made it implement "Comparable," so that an array filled with Objects of the class Psg would be able to be sorted by score. In a nutshell, I first fill an FSArray structure that holds Passages and that is a feature of my type "QASet" with the unsorted passages corresponding to a question (as in the previous assignment). Then I pass those passages through the Psg class, so to speak, into a Psg array, and sort that; then they are ordered and can be put back into a new, sorted, "RankedPassageFSArray" feature of the QASet type.

It occurred to me after I implemented this that since the "Passage" is already a regular Java class as well as being a UIMA type, I probably ought to have been able to implement "Comparable" in `Passage.java` and saved myself some trouble. But I didn't do that, and what I have is working, if less efficient than it could be. Maybe for next time.

Using that data structure and the rest I have established, writing the code to calculate the metrics was a simple matter. I was at first unsure exactly what was being asked to be returned for reciprocal rank and precision at $n > 1$, so at first I wrote code that was more complicated than necessary (can be seen commented out in `InputDocumentAnnotator.java`). But I asked about these on Piazza and got answers to my questions, so fixed those. Printing the results using the CAS consumer ("PassageRankingWriter.java") to a .csv file was also simple. I use standard out to print mean reciprocal rank and mean standard precision; those final values are:

Mean average precision: approx. 0.35790
Mean reciprocal rank: approx. 0.39435

In comparing these values to those of one of my classmates, these are both lower than the ones returned by her system. The reason for this is probably that (as I described in last week's report), I am using an F-score over unigrams as the score by which passages are ranked, and not just unigram precision. I would have assumed that my method would likely yield better results, because one tends to think that using both precision and recall is more sophisticated and hence more accurate than just using precision. But it's worse! That's a good lesson. And it underlines the usefulness of having these metrics by which to compare different scoring methods, obviously.

I am not an IR or IE person, when it comes to my main studies here at the LTI. But I feel that a couple of the metrics we're asked to implement here are not defined the same way as my intuition tells me they "should" be. Doubtless this is because I am inexperienced in this subfield. But, regardless, my feeling is that, speaking as a human and not a machine learning system anyway, the "P@n" metric is not that useful when the number of correct passages is significantly less than the value of n. If there are only two correct passage out of 12 total passages, then the highest score possible for $n = 5$ is 0.4. To me, this looks low and doesn't indicate that in fact, that score in that context is the highest possible score. You have to look at the total number of correct answers to glean that knowledge, which is

an extra step that I wouldn't want to have to do. I would want to have the denominator be limited to the total number of correct answers. On the other hand, I suppose that having the higher measurements for this metric be reserved for those questions that both have many correct answers and rank them highly makes some sense too. I'd want to use both these methods, actually. No doubt machine learning methods in this field make use of all of these relatively simple metrics that they can; and perhaps empirical results over the years have indicated that certain ones tend to be more useful than others.

Along the same lines, the reciprocal rank metric as defined is fine, but I prefer finding the reciprocal rank of each correct passage in a set and averaging those, for a mean reciprocal rank measurement defined over each set. Then this metric could also be averaged over the whole dataset, resulting in one mean-of-means value returned at the end. Again, this would be one more datapoint that I don't know whether would be more or less useful than the ones we are returning in this assignment; but I would want to see it.

Finally, why are we reporting average precision over all the passages in a set but just returning precision at 1 and precision at 5? I would want to also return precision at *n* where *n* is the total number of passages, as well as to return average precision at 5, since we're doing precision at 5. In these ways, the numbers could tell more of the whole story. Of course, the numbers are only as useful at distinguishing between good and bad question-passage sets as the scoring function is at its job, namely determining how good or bad a passage is at answering its question. So to get more out of these metrics, I'd want to try other scoring methodologies.

The feature I like best about UIMA thus far is the Document Analyzer run configuration. The visual interface for viewing often-overlapping annotations and their spans and their features and the features' values is very useful for debugging and also just for data visualization itself. Both the span viewer on the left and the hierarchical view on the right make it as easy as I think it can be in a task like this to comprehend the data—which passages are linked to which questions (and back again), and also seeing the output values of the metrics we calculated all in one place, which allows for debugging one's calculations if something looks off. Of course, it would not have been that useful if I had not defined my type system to give such a useful result! I think that is the first time I've really been able to see how by planning ahead and using this complicated, often

redundant-seeming framework, one can actually receive a useful payoff in terms of comprehensibility.