11-791: Design and Engineering of Intelligent Information Systems

Project Individual 7: Architecture & Design Patterns

Keith Maki (andrewID: kmaki)

October 19, 2015

1 Introduction

In this report, I present my design and implementation for the aggregate passage ranking model pursuant to the PI7 sample information processing system design task of developing a flexible framework for supporting a passage ranking information retrieval system. The report is structured as follows: section 2 describes the overall framework pipeline; section 3 provides discussion of important design considerations and limitations to the selected approach; section 4 presents the relevant project files and data; and section 5 concludes the report.

2 Passage Ranking System Design Overview

I present a generalized framework to support an aggregated passage ranker. I extend the passage ranker from the previous Project Individual, which took an ngram-based approach to computing the ranking, but expand the flexibility and extensibility so that it is easy to exchange this ranking approach with an arbitrary passage rankers, including Composite pattern rankers. I will focus in this section primarily on the pipeline flow and development process for the system presented. For specific design considerations, including a thorough discussion of the design patterns, see section 3.

The UIMA framework makes it easy to sequentially chain arbitrarily many analysis engines to process documents in a modular way, building new annotations with the help of those from previous layers. Following the previous Project Individuals, I implemented a series of analysis engines which are chained together to form the passage ranking system. The analysis framework is described in detail below. The primary changes to the passage ranking system to support extensibility and flexibility were implemented in the Scoring phase, however there have been several additional changes as well. For ease of readability, components which are not changed significantly from the previous PI(s) are demarcated with the tag **No change**.

```
1894 QUESTION How far is it from Earth to Mars?
1894 APW19980923.1395 1 hours, JPL said. After five more months of aerobraking each orbit should take less than
    two hours.  Mars is currently 213 million miles (343 million kilometers) from Earth.
1894 APW19990429.0168 -1 ‘‘The data suggests that Mars was once magnetic and was far more similar to Earth’s
    global magnetic field than had been previously assumed,’’ Ness said.
1894 APW19990429.0249 -1 ‘‘The data suggests that Mars was once magnetic and was far more similar to Earth’s
    global magnetic field than had previously assumed,’’ Ness said.
1894 NYT19981211.0308 -1 c.1998 N.Y. Times News Service <TEXT> <P> CAPE CANAVERAL, Fla. _ The first of two NASA
    science probes embarked Friday on a 416-million-mile voyage to Mars, catapulted from Earth by a Boeing Delta
     2 rocket. </P> <P> The spacecraft, called Mar
1894 NYT19990923.0365 1 its farthest point in orbit, it is 249 million miles from Earth. And, so far as anyone
    knows, there isn’t a McDonalds restaurant on the place. </P> <P> And yet we keep trying to get there. Thirty
     times in the past 40 years, man has sent a spacecra
1894 APW19990429.0168 -1 LINE> <TEXT> <P>  WASHINGTON (AP) -- Early in its history, the geology of Mars may have
     been much like that of Earth, with molten rock gurgling to the surface from a hot interior and with crustal
     plates drifting on the surface, according to new s
1894 NYT19990923.0363 -1 so see Mars-sider) By MARK CARREAU c.1999 Houston Chronicle <TEXT> <P> After a trouble-
    free, 91/2-month voyage from Earth, NASA’s Mars Climate Orbiter disappeared early Thursday as it was
    maneuvering into orbit around the Red Planet. </P> <
1894 NYT20000324.0337 -1 For more than two years, planetary geologist Ken Edgett has been among the first
    experts to examine the steady stream of photography dispatched to Earth by NASA’s Mars Global Surveyor, a
    small camera-equipped probe that has been circling the Red Plan
1894 NYT20000324.0337 -1 geologist Ken Edgett has been among the first experts to examine the steady stream of
    photography dispatched to Earth by NASA’s Mars Global Surveyor, a small camera-equipped probe that has been
    circling the Red Planet since September 1997. </P> <P>
1894 NYT19990313.0104 -1 far. </P> <P> The rain of debris from Mars was hardest, experts agree, in the Earth’s
    early days. And the reverse trip was far less likely because the Sun pulls Earth debris away from Mars and
    toward itself. </P> <P> Burns of Cornell and his
```

Figure 1: Example input question with ten passage choices.

1. Document Reader **No change**. This preliminary stage manages the input to the system, reading in questions and passages of the form presented in Figure 1. The questions and passages are not assumed to be in any order, and no assumptions are made about the number of passages or questions present in the input document. The document is assumed to contain at most one question per question id, but the question ids need not be sequential (or even integral). The use of an initialization method in my implementation of this class allows for the input document to be sorted into separate question-passage entries. These entries are then used to populate the CAS when the document reader's getNext() method is called.

2. Aggregate Analysis Engine

   2.1. Analysis AAE

      i. Test Element Annotator **No change**. In this stage, the document is parsed according to the major sections visible in the example input in figure 1. The question(s) and corresponding passage(s) present in the input document are identified and instantiated as Questions or Passages using the defined UIMA Type System. Additionally, appropriate annotational meta data is identified and recorded (e.g. for a passage, this includes the source document id, the label identifying whether the passage correctly answers the question, the isolated text of the passage, and the indices into the text which identify the raw input which is being annotated.). Copies of the raw spans for the questions and passages are also created which are then maintained for use in the downstream annotators and CAS Consumers.

      ii. Tokenization Annotator **No change**. In this stage, each portion of the document (question and answers) is tokenized using Java's StringTokenizer utility class. Each token is given a separate annotation span, and the collections of tokens from each of the portions described above are given collective annotations. Each question in the document is additionally given a single, separate TokenAnnotation which groups its tokenized span and the tokenizations of each of its answer choices. A pointer to the raw input of each tokenization is also preserved, for the benefit of any downstream tasks.

      iii. N-gram Annotator **No change**. In this stage, each of the tokenized spans from the previous annotator are annotated with ngrams of fixed (parameterizable) length. The parameter is read from the UimaContext when the annotator is initialized. Again, pointers to the raw input spans of each ngram set are preserved for the benefit of downstream tasks.

      iv. Score Annotator In this stage, each passage is assigned a similarity score using its annotations and those of its respective question that have thus far been placed in the CAS. In contrast to the previous assignments, the similarity score is handled using a generic ranking framework (see package "rank" in section 4). This framework allows for the instantiation of configurable ranker classes to facilitate the implementation of more intelligent

and involved system design strategies. As was the case for previous assignments, one of the submitted configurations implements a basic similarity score which is computed based on percentage ngram overlap. Next, a second similarity metric was implemented, which used the longest case-insensitive substring between the question and passage texts. Additionally, a Weighted Average Composite ranking strategy was also implemented, which allows for the weighted average of the scores from a combination of individual ranker outputs to inform the passage ranking. Design considerations, including implemented design patterns, are discussed in 3.

2.2. Evaluation AAE

i. Error Analysis Annotator **No change** This annotator computes a number of metrics with which systems may be compared. As was the case last week, there seems to be a bug in this system, which will necessitate additional debugging. This was not prioritized due to the focus this week on improving the extensibility of the ranking strategy framework.

ii. Output Annotator **No change** In this final analysis stage, the evaluations for each question are gathered and compiled into an output string matching the format of the task.

3. CAS Consumer **No change** The final stage of the pipeline manages the system output, writing the output annotations to a single output file.

3 Design Considerations

[1] This week presented a very exciting opportunity to make use of design patterns to improve the usability and extensibility of the design framework which I have been devloping for the better part of two months. I took advantage of this chance to apply new knowledge and better my understanding of their application within the domain presented by this passage ranking task. However, I was careful to understand the underlying justifications for and ramifications of the design patterns used.

- The IRanker interface which was provided in the template code allowed for the development of different **Strategies** which could be used interchangibly through a shared interface. This interface was accessed by the ScoreAnnotator class, but built upon and extended by the various implementing classes.

- In designing a ranker to perform a weighted average of other ranker outputs, I first made use of the **Composite** pattern, taking advantage of the IRanker interface once more to treat rankers the same way through their shared interface. I developed an abstract class, CompositeRanker, which extends the provided class AbstractRanker (and thus implemented IRanker), to scaffold this composition, and then I extended

[1] Unless otherwise indicated, all classes mentioned in this section are part of the rank package. See 4 for more information.

it with a concrete class, WeightedAverageCompositeRanker, to perform the specific type of averaging desired for this assignment.

- I then applied the **Bridge** pattern to the scoring algorithm portion of each ranker, beginning with the AbstractRanker class. I designed an interface, ScoringAPI, which provides a conceptual level of a Scoring algorithm and describes how to apply it to score a question-passage pair. This interface allowed me to concretely describe the method for scoring a question-passage pair inside AbstractRanker, effectively decoupling the extending subclasses from the ScoringAPI interface class describing the method of scoring. This helps to keep the classes implementing ScoringAPI focused, organized, and reusable, but also enables greater specificity in the AbstractRanker level and ease of implementation at the concrete Ranker level.

- Similarly, I applied the **Bridge** pattern to the CompositeRanker class, enusring the abstract description of a composite ranker could describe how to compose scores from component rankers without restricting the method of composition. I was impressed at how much more streamlined these patterns made the development of new ranking implementations, as it was fairly straightforward to develop the other concrete classes with the organizational and inheritance-related benefits inherent in practical design making use of this pattern.

- Additionally, I implemented the **Builder** pattern, this time, working at the interface level. I modified the IRanker interface to include a static interface, IRankerBuilder, which defined a method for instantiating an IRanker. Appropriate use of this interface allows for creation of IRankerBuilder classes which ensure low interdependence between downstream/application code and the underlying IRanker implementions. Additionally, instantiating multiple IRanker instances with the same or similar configurations can be done cleanly and simply by requesting additional instances be produced by a single IRankerBuilder instance.

- The **Iterator** pattern was not directly included in the scoring and passage ranking framework I developed for this assignment, but I did make use of an Iterator pattern application in my util package classes. The ListReverser class definees an implementation of Java's Iterable interface which iterates through the elements in an Iterable in reverse order. The use of the Iterable interface by this class ensures that the desired iteration may happen regardless of the underlying representation of the elements (indeed, in this case the elements would normally be accessed in the opposite order).

Although I had good success in concretely and elegantly implementing the presented design patterns in my passage ranking framework, I did deprioritize the ranking framework performance as a whole. As was true last week, one major limitation in the current evaluation engine framework is that the evaluation metrics are all implemented within a single primitive analysis engine. Making use of techniques similar to those introduced in the scoring components this week could greatly improve the design of the evaluation framework.

I still find it very challenging to divide my efforts among several modules while working under a time crunch, and this week found me again unable to fully analyze the performance of my PI within the allotted time. Although I can only budget so much time to spend on the assignments for this course, I have begun to build up a repertoire of analysis engines and type system components which are flexible enough to be applied in fairly general projects. Ultimately, however, development of these systems is still time consuming, and I spent a significant amount of time this week chasing the aforementioned error in the System Analysis portion of the code. While my system does compute precision, recall, and f1-scores for each test element it processes, the output continues to be incorrect.

Once again, I have a fully usable system this week, which can be installed and executed through Maven, but since it took a significant amount of time to implement the design patterns and other improvements in the passage ranking portion of the pipeline, the concrete implementations and configurations instantiated by the submitted system are ultimately fairly trivial.

Somewhat unfortunately, though, the system lacks the ability to be fully configurable through descriptors, in that configuration parameters for particular Rankers currently must be hard-coded into the ScoreAnnotator file as options to be selected using a UIMAContext parameter setting. In order to fully integrate this ranking framework into the descriptor file interface, an instantiation of the Factory pattern or a similarly generalizable approach would need to be implemented which could aggregate the desired parameter values for which IRankerBuilder implementation should be instantiated and how to configure it.

4 Architecture and Design Patterns UIMA CPE Implementation and Results

The full pi7-kmaki project repository including this report can be obtained at https://github.com/kortemaki/pi7-kmaki.

5 Conclusion

In this report, I presented my generalized UIMA passage ranking framework in pursuit of the requirements of the PI7 assignment. The techniques employed in the design of this system demonstrate the powerful nature of the design patterns we have studied in this class up to this point. I found the usefulness of these patterns was really only appreciated once I solidified my understanding through the implementation of this assignment. However, while these systems come with a greater degree of extensibility and power, their implementation is certainly more time consuming than "throw-away" procedural implementations. In the development of significantly more elaborate systems than can realistically be written in a single procedure, however, these tradeoffs will cease to play a role, and the proper use of appropriate design patterns will become essential to scaffolding effective software design.