Andrew Wilkinson
DEIIS Fall 2015

"Project Individual" 7 Report: Pigs, Hogs, Sows, and Boars (not really)

I'm writing this a little hurriedly because just when everything was working fine, for some reason that I can't fathom based on something I must have done (but I don't know what I did), Eclipse decided that the methods in the project that override methods in superclasses do not actually override methods in the superclasses. So despite the fact that I compiled and ran it without incident plenty of times up till now, there are several lines that are giving errors. The line `public void processCas(CAS arg0) throws ResourceProcessException` in PassageRankingWriter.java tells me, `The method processCas(CAS) of type PassageRankingWriter must override a superclass method.` The same thing goes for `public Double aggregateScores(List<Double> scores)` in CompositeRanker.java. I can run `mvn install` but I can't run the program anymore; I get this error in the terminal:

```
[ERROR] Failed to execute goal
org.codehaus.mojo:exec-maven-plugin:1.4.0:java (default-cli) on
project pi7-andrewid: An exception occured while executing the Java
class. Unresolved compilation problem:
[ERROR] The method collectionProcessComplete() of type
StatusCallbackListenerImpl must override a superclass method
```

The stupid thing is that I had that same error plaguing me yesterday, and it went away on its own, without my having to have changed anything. So I promise that I was running my project merrily just half an hour ago. The output file is, at least, in the output directory.

UPDATE: Okay, I may have fixed that. But now it's giving me a problem where it says it has finished running, but my output file is blank; I think the problem is that I am using Meteor as my "OtherRanker," but had some kind of difficulty adding that to the build path as a permanent thing, and not just an external jar that I'm linking to.

```
ThreadGroup.uncaughtException()-Got Error
Oct 19, 2015 11:29:07 PM
org.apache.uima.collection.impl.cpm.engine.CPMThreadGroup process
SEVERE: The CPM thread group caught the following unhandled error:
java.lang.NoClassDefFoundError (Thread Name: [CasConsumer Pipeline
Thread]::)
```

Well, that can be fixed, but I don't have time to right now. It runs using the CPE GUI run configuration, anyway! I replaced the output file with the good version again. I'll just go on and describe what I did for the project.

The bulk of my time was spent transferring everything I did for the previous project to the template and making it all still work. I experimented with several ideas for the "OtherRanker" besides my n-gram ranker (which uses an *n* of 2), and decided to try running the Meteor MT evaluation metric on the question and passages to output a score. Implementing this was not difficult, since Meteor is a Java program and I could use its basic API; I just had to pass the Meteor scorer through the pipeline of this project as a parameter. I have significant experience using Meteor for MT purposes. Last semester I did a lab in which I extended its functionality to use POS classes instead of just a content/function word distinction. So I am very familiar with how it works and what the scores it outputs represent.

Those scores essentially are all about recall and precision, based on unigrams, which is the same basic thing as the tokens and n-gram approaches we've been using in this class. But Meteor is more sophisticated than what we've implemented, because not only does it use basic (normalized by stemming and lowercasing) tokens, but it also gives some weight to matches between tokens based on synonyms and paraphrases. I thought this could be an ideal way to implement matching based on these types of relationships between words and phrases in our sentences without having to write it all myself. Furthermore, I included a normalization process by which a potential reduction in score based on the length of the passages in tokens is eliminated. My thinking was to remove a bias either for or against verbose sentences, depending on context (passages closest to the length of the question would otherwise be likely to receive higher scores overall), and concentrate on semantic

similarity. Perhaps the passages with the highest similarity to the questions, along the dimensions Meteor measures, would be more likely to be correct matches.

As it turned out, this method did not do as well as my bigram method. But it was worth a shot. I tried different values for the weighted average besides 0.5 for each ranker, and of course the higher the score on the better of the two rankers was, relative to the other, the better the overall F1 scores and other measurements became. I settled on a 0.8 / 0.2 distinction for my output file, just to show off what was happening in a somewhat balanced way. There was one improvement, in one statistic, when I combined the two rankers, and that is in MRR: with bigrams, it was 0.595; with Meteor, it was 0.495; with the composite ranker it was 0.645. This shows me that analyzing gains from a composite-ranking process is a nuanced business, and it doesn't come down only to one or two measurements. It also shows that adding a ranker that does less well by some metrics can still improve other metrics.

I don't have time now to describe the design patterns I used besides what was there in the template. That's pretty much what I relied on. I think that having the two rankers both extending AbstractRanker is a composite pattern, and that having CompositeRanker implement IAggregator is a bridge pattern.

Okay, let's turn this in! Thanks for reading!