PI7 - Report

Yash Gaur (ygaur@andrew.cmu.edu)

1. Identify design patterns implemented in the template project if you use it. Describe all the design patterns that you implemented, including those in the template project, showing some class diagrams when appropriate (5 pts).

Composite Pattern was implemented in the way *compositeRanker* and *ngramRanker/otherRanker* were implemented. The composite ranker, which is basically a list of individual ranker, implements the same interface as individual rankers. This allows us to use the composite ranker in the same way as any other ranker, making it easy for us to write the CAS consumer.

Bridge Pattern was implemented in the project template when there was a 2 level abstraction/inheritance between the interface IRanker and concrete implementations like NgramRanker. The purpose of the Bridge pattern is to decouple an abstraction from its implementation, which has been implemented as the 2 level abstraction between IRanker, AbstractRanker and NgramRanker/OtherRanker.

Builder Pattern was used when I was implementing my ngram ranker class. There are certain parameters for the class, like the degrees of ngrams that could be extracted, for which we may have to write a number of constructors, leading to the telescoping constructor anti pattern. To avoid this I made a public builder class within the ngram class. This class can be initiated with degrees of the ngrams that you want to extract. This class will then build the ngramRanker class using the parameters that were passed to it.

Iterator Pattern was implemented using the java's Iterator interface. Every instance when we need to iterate over a list (like List<Passage>), I used the iterator as below:

```
for (Iterator<Passage> iter = passages.iterator(); iter.hasNext(); ) {

Passage passage = iter.next();

// do stuff with passage

}
```

2. Explain justifiable reasons why you implemented those design patterns (5 pts).

Composite pattern was implemented because the compositeRanker needs to be treated in the same way as any individual ranker. This composite ranker is essentially a list of individual rankers and hence a composite pattern, where both the individual object and the composite object implement a common interface, makes sense.

Bridge Pattern was implemented because there is a great variety of ranker that can be extended from the abstract class for the rankers. To attain flexibility, decoupling is required between the abstraction and its implementation. This a Bridge Pattern makes sense.

Builder Pattern was implemented because there can be many parameters that are passed to the NgramRanker class. If we write constructors for every combination of parameters, then we have exponential growth in the number of constructors, leading to telescoping constructor anti pattern. Thus making a builder class makes sense.

Iterator pattern makes a lot of sense as we have many instances where we have to iterate over Lists. Doing a simple for loop and getting an element using the get method is not efficient since it does not store a point from which it needs to read for getting the next element. Iterators solves this problem and the code becomes a lot efficient when reading/working with big lists.

3. Describe what ranking strategy you chose to implement besides the N-gram ranking strategy (5 pts).

The alternate ranking strategy used is a slightly modified implementation of tf-idf ranking technique. This technique is implemented in the *otherRanker* class. For each question and associated passages, we first calculate the idf scores for all the words in the question after the stop words have been removed. These idf scores are calculated based on the passages that are associated with the questions. In the second step, for each passage and each question word, the term frequencies are calculated. The way term frequencies are calculated is different since here, we count the total number of occurrences of a term in a passage, and then divide it by the count of a question word that has the maximum occurrences in that passage. For example:

Question word	P1(tf-count)	P2(tf-count)	IDF
QW1	2	3	0.23
QW2	1	2	0.65
QW3	0	1	0.12

After normalization:

Question word	P1(tf-count)	P2(tf-count)	IDF
QW1	1	1	0.23
QW2	0.5	0.66	0.65
QW3	0	0.33	0.12

The tf-idf vectors are calculated by getting the dot product of the normalized tf counts and the IDF vector. The similarity between between a passage and a question was implemented using cosine similarities between the tf-idf vectors of the question and the passage.

4. Explain justifiable reason(s) why you chose to implement the additional ranking strategy (5 pts).

The reason why one can imagine this to work is because passage ranking is quite similar to document ranking in many ways. As with query and documents, question-answering and passage ranking can also said to be highly dependent on some important words. Tf-idf tells us how important a word is in a particular document and also provides a means to get vector representation for every question and passage. This vector representation can be used to calculate similarity, which can in turn help us in ranking the passages.

5. Describe your error analysis on the composite model, focusing on the difference between the composite model and the N-gram ranker/your additional ranker (10 pts).

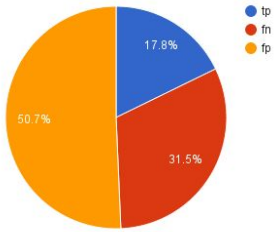
Error Metrics:

Techniques	Ngram Overlap	modified tf-idf	composite
Micro F1 Average	0.2188	0.2634	0.2188
Macro F1 Average	0.3023	0.3488	0.3023

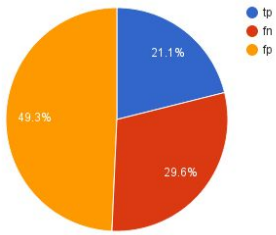
Error Analysis:

We see that the modified tf-idf technique works better than the Ngram overlap score. However, the composite ranker's performance is equal to the worse of the two. Below are the pie charts for each technique:

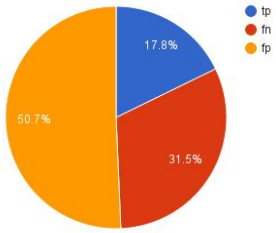
Ngram Approach:



TFIDF approach:



Composite technique:



There are a high number of false positives in all the cases. This can be attributed to the way in which the system assigns true and false labels. Currently, the top 5 ranked passages are assigned true by the system. This is clearly a bad idea as the average number correct passages per each question is clearly lesser (somewhere between 1 and 2 I think). The TF-IDF approach gives more true positives and lesser false negatives and false positives. The reason I think it performs better is because, the questions in the dataset are factoid questions. Ranking passages is very similar to the document retrieval/ranking tasks used in search engines.

The composite technique seems to be working worse than TFIDF in the subset of questions I am working with, however, I did encounter subsets (with other random seeds), where it worked at par with TFIDF. I tried various weights for the weighted average in compositeRanker, all of which gave similar results. The result I am reporting are for equal weights (0.5, 0.5). A random seed was used to sample a subset of questions.

NOTE: I was able to build and run the project using maven

commands:

mvn clean

mvn install

mvn exec:java -Dexec.mainClass="Main" -Dexec.args="src/main/resources/inputData src/main/resources/outputData"