PI4: COMPONENT AND DESIGN PATTERNS

Name: Zhuyun Dai
AndrewId: zhuyund

---

1. OVERALL STRUCTURE

The task of the PI4 is to implement a Question-Answering System with NIST TREC 2003 Question Answering data.

This system is designed for the TREC 2003 QA data. Passages in the dataset were crawled from the web, thus contains noise such as html tags and non-alphanumeric characters. In order to improve the quality, the system cleans the data by removing html tags and stop words.

In the passage ranking part, I used bag-of-words model to represent question and passages. Bag-of-words model preserves most of the information, and easy to implement. 2 methods are used for scoring: Boolean match and cosine similarity. The user can specify in the parameter which method to use. The Boolean Match is set to be the default method.

In my implementation, the Collection Processing Engine (CPE) contains 3 parts: a Collection Reader that reads the question and passages from the input directory; an Aggregated Analysis

Engine that annotates the document and scores each passage; and a CAS Consumer that ranks the passages and writes the ranked list into the specified file.

The Collection Reader reads the data collection into the CAS. The data is a file containing questions and passages.

The Aggregated Analysis Engine is made up of 3 primitive AEs. All AEs extend JCasAnnotator_ImplBase.

- `TestElementAnnotator`. Annotates question and passages in the document. It records the id and text of question/passages. It also records which question a passage is related to.
- `TokenAnnotator`. Cleans and tokenizes question and passages.
    - First, it casts all text into lower case.
    - Second, html tags such as <P> are removed from the text.
    - Third, it uses `stanford.nlp.process.Tokenizer` to tokenize the text.
    - Finally, stop words such as 'a' 'what' are removed from the text.
- `ScoreAnnotator`. Scores each answer by the cosine similarity of bag-of-words models of question and passages. It takes a parameter "`METHOD`" to decide which scoring method will be used: Boolean Match or Cosine Similarity.

The CAS Consumer is the final part of the pipeline. It sort the passages by their question Id and score. The ranked list is written into the output directory specified in the command parameters.

---

2. DESIGN PATTERN ANALYSIS

In the system implementation, 2 design pattern are used: the Factory Pattern and the Template Pattern.

2.1 Factory Pattern: FSListFactory

In the system, many features are made up of a list of features. For example, a passage contains a list of tokens. UIMA's type `FSList and StringList` are often used to stored these features. However, when processing data, we often use Java's Collection to store intermediate results. For example, in the TokenAnnotator, tokens are stored into `ArrayList<String>`, and passed to the function `removeStopWords()`. UIMA does not provide APIs that easily convert Java Collection to FSList (or StringList, FloatList,…). Therefore, we need a class that bridges between JAVA colletion and UIMA lists.

The Factory Pattern is one of the creation patterns. A factory creates object without exposing the creation logic to the client and refer to newly created object using a common interface. `FSListFactory` creates Java Collections from UIMA lists. It also creates UIMA lists from Java Collections.

`FSListFactory` has 4 functions:

- `createCollection(FSList, Type)`
- `createCollection(StringList, Type)`
- `createFSList(JCas, Collection<? extends ComponentAnnotation>)`
- `createStringList(JCas, Collection<String>)`

I did not implement methods for FloatList/IntegerList because the system only uses FSList and StringList. When creating Java Collection, the user does not have to specify the type of collection being created.

2.2 Template Pattern: Scoring

In the `BooleanMatchScoring.evaluateScore`, 2 scoring methods can be used: Boolean Match and Cosine Similarity of term frequencies. Boolean Match counts the token overlap between the question and the passage. Cosine Similarity computes the cosine value of the TF (term frequency) vectors of the question and the passage.

3

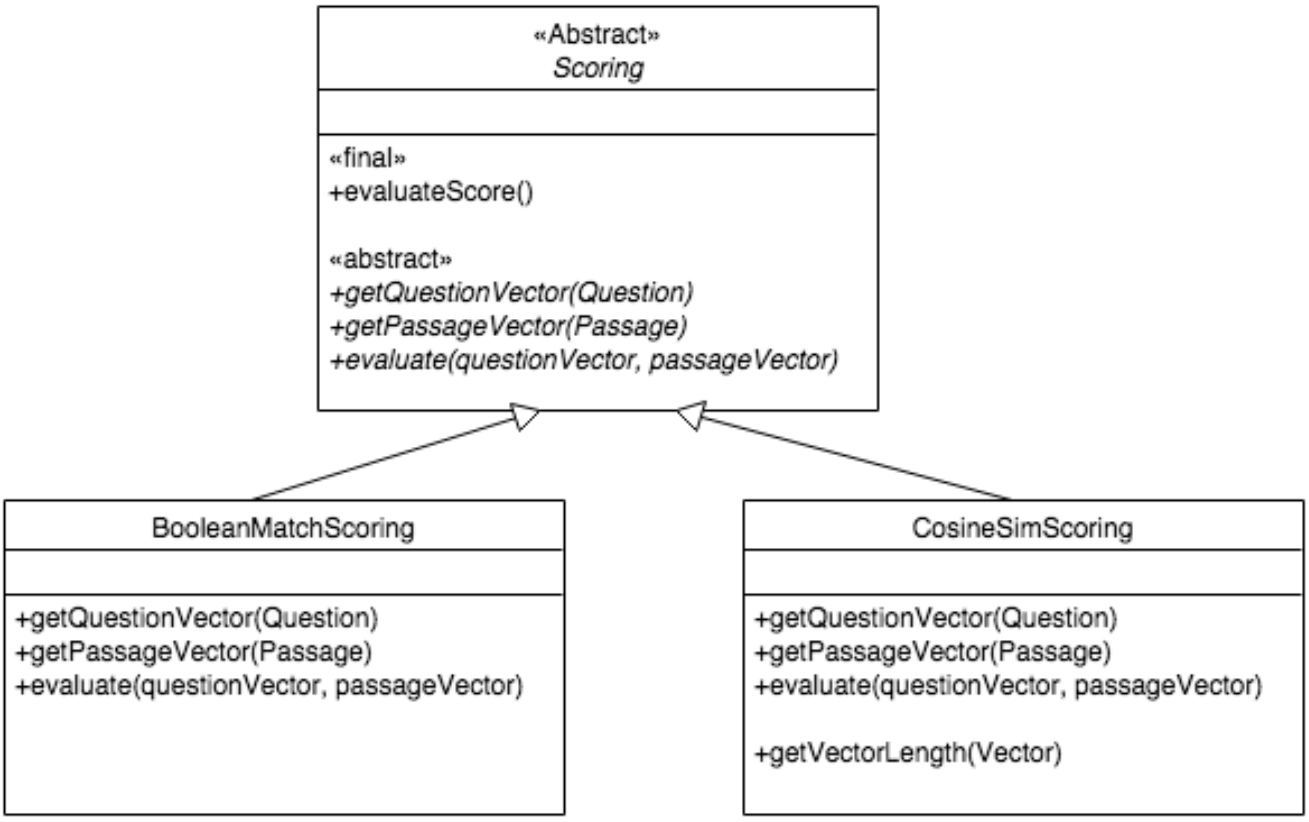
Both methods use the same logic:

1. Compute the vector representation of the question and the passage.
2. Compute the score based on the vectors.

However, they are different in the details.

1. Boolean Match uses 0/1 vectors to represent text. Cosine Similarity uses the TF.
2. Boolean Match uses the inner product of the vectors as the score. Cosine Similarity needs to normalized the inner product by the vectors' lengths.

Template Pattern defines the skeleton of an algorithm in a method (the template method). This template method shows the steps of an algorithm. Steps are abstract and need to be implemented by subclasses. Template Method lets subclasses redefine certain steps of an algorithm without changing the algorithms structure. The UML graph of the Scoring is:



`Scoring` is the template class. The final method `evaluateScore()` defines steps for the scoring: it calls the abstract functions `getQuestionVector(), getPassageVector()` and `evaluate()`.`BooleanMatchScoring` and `ConsineSimScoring` implemented those abstract functions with respect to their own needs.

In the AE `ScoreAnnotator,` it calls the `BooleanMatchScoring.evaluateScore()` or `CosinesimScoring.evaluateScore()` according to the parameter "METHOD".

---

3. SCORING METHODS

2 scoring methods are implemented and used in the system: Boolean Match and Cosine Similarity.

Boolean Match represents each document as a 0/1 vector: 1 if the token appears in the document. The score is the inner product of 2 vectors. That is the # of overlapping tokens in the question and the passage.

Cosine Similarity represents each document as a integer vector: each element in the vector represent a term frequency. The score is the cosine value of 2 vectors.

In the `PassageRankingWriter,` the macro-average Precision@N $AP = \frac{1}{\#\ of\ question}\sum_q P@N_q$ is computed. N_q is the number of correct answers of question q.

Method	**AP**
Boolean	0.3147
Cosine	0.2644

Boolean Match performed better than Cosine Similarity.