Homework 4

Salvador Medina
139323
DEIIS, ITAM

April 26, 2014

1 Introduction

In this homework we are required to complete a Retrieval analyzer built on a UIMA aggregate analysis engine. The AAE's input are single documents formed by a single sentence. All the documents are read from the same file *documents.txt* found in the resources folder in the given template project. The main tasks for this homework can be summarized as follows:

1. Genrate the type system implemetation using UIMA's JCasGen
2. Extract the bag of words feature vector from the input text collection
3. Compute the *cosine similarity* between two sentences
4. Compute the *mean reciprocal rank* (MRR) for all the senteces per retrieval
5. Design a better retrieval system that improves the MRR metric
6. Improve efficiency of the program by doing an error analysis of the retrieval system

1.1 AAE

The given AAE main goal is to read a set of documents merged into a file, parse a set of retrievals formed by a query and answer, find the best suitable answer to the query and calculate the MRR of the scoring system. It is composed of three annotators:

1. Document Reader
2. Document Vector Annotator
3. Retrieval Evaluator

As shown in Figure 1

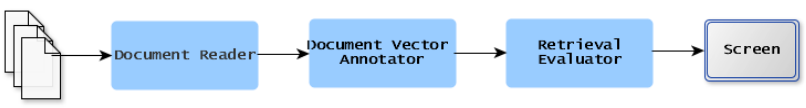

Figure 1: AAE

The *Document Reader* annotator is in charge of reading the input file, parsing each line and creating the *Document* annotations storing the text, qid *(query ID)* and its relevance value. The relevance value indicates the type of sentence we are dealing with. The values interpretation are:

- 99 : it is a query
- 1 : it is an incorrect answer
- 0 : it is a correct answer

The *Document Vector* annotator tokenizes the single sentence documents. Our tokenizing process is quite naive as it takes the text out the *Document* annotations, removes the punctuation marks, separates the words from the sentences by whitespaces and creates a *Token* annotation per word, builds an *FSList<Token>* array which is set to the corresponding *Document* annotation.

Finally, the *Retrieval Evaluator* is the most important one as it is in charge of analyzing the whole preprocessed data. Once the documents are correctly annotated, it is able computes a certain similarity between sentence and give a MRR score to the answer detection system.

2 Tokenizer

To start with, we created a naive tokenizer that takes the text out the *Document* annotations formed in the *Document Reader* annotator. Then it removes the punctuation marks, separates the words from the sentences by whitespaces and creates a *Token* annotation per word. With the extracted tokens the annotator builds a *FSList<Token>* array which is set to the corresponding *Document* annotation.

3 Retrieval

In this particular problem, the most important element is a retrieval. The retrieval is the elment which we want measure with the MRR. A Retrieval is described by a query and its answers. These elements are given per document, however the elements of a particular retrieval are related by a *qid*. Therefore, we thought it would be convenient to create classes which contain these elements. Their description is easier to comprehend by looking at they UML diagram in Figure 2.

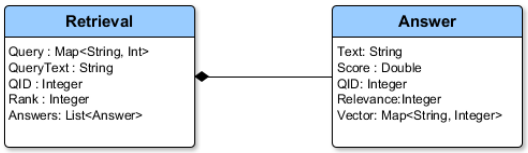

Figure 2: New Classes

As all the documents are tokenized into words and stored in a global dictionary. In such a way that each query or answer is described by a vector. This vector tells which of the words from the global dictionary are located to its corresponding sentece and the frequency of the word.

However, to store these vectors in a static way would consume a lot of memory, as each position in the array corresponds to a particular word from the global dictionary. It is also known that in most of the cases, the vectors are sparse as they only use a small fraction of the words from the global dictionary. For this reason, it felt convinient to use a *HashMap* to store the sparse vector, where the *key* would be the word used and the *value* is its frequency.

4 Scoring

Once we have the retrieval wrapped up into an object. It becomes easier to handle the data and make operations over it. In the original requisites of the problem to score each answer according to its query, it is required to so by calculating the cosine distance based on the words of each sentence.

4.1 Cosine Distance

The cosine distance was implemented in the *computeCosineSimilarity(...)* method which makes use of other two auxiliary methods: *computeDotProduct(...)* and *getMagnitude(...)*. The first accepts two sparse vectors (HasMap's) and calculates the similarity according to:

$cos\theta = \frac{A \cdot B}{|A| x |B|}$

The dot product is computed by the second method which accepts two sparse vectors. The prodcut is calculated by accumulating the multiplication of the frequency of the common words to both vectors.

The magnitude of each sparse vector is calculated by the third method by obtaining the square root of the sum of all the word frequencies eleveted to the power of 2.

Once all this is calculated, the score is saved in its corresponding *Answer* object.

4.2 Giving an order

To order the *Answer*'s of each *Retrieval* we took advantage of Java's *Collections.sort()* method for this reason, the *Answer* class was implemented based on the *Comparable* class and the *compareTo* method was adapted to sort them according to its score in a descending form.

Once the answers were sorted according to their scores, the rank was easily obtained by searching for the first answer in the list of answers which has a relevance equals to 1. The rank is saved in the *Rank* attribute of the corresponding *Retrieval* instance.

Finally the MRR was easily computed accordin to the ranking obtaind in each retrieval.

5 Results

The results obtained were satisfactory as by removing the punctuation marks, transforming all words into lowercase, removing the stopwords and calculating the cosine distance we obtained the results described in handout paper. Obtaining an output as follows:

```
Query:
   QID:  1
   Text: Classical music is dying
   Rank: 1

Answers:
   Rel: 1  Score: 0.61237  Classical music may never be the most popular music
   Rel: 0  Score: 0.51640  Everybody knows classical music when they hear it
   Rel: 0  Score: 0.38490  Pop music has absorbed influences from most other genres of pop

Query:
   QID:  2
   Text: Energy plays an important role in climate change
   Rank: 1

Answers:
   Rel: 1  Score: 0.46291  Climate change and energy use are two sides of the same coin.
   Rel: 0  Score: 0.00000  Old wine and friends improve with age
   Rel: 0  Score: 0.00000  With clothes the new are the best, with friends the old are the

Query:
   QID:  3
   Text: One's best friend is oneself
   Rank: 2

Answers:
   Rel: 0  Score: 0.56695  My best friend is the one who brings out the best in me
   Rel: 1  Score: 0.50000  The best mirror is an old friend
   Rel: 0  Score: 0.25000  The best antiques are old friends

 (MRR) Mean Reciprocal Rank ::0.8333333333333333

Total time taken: 1.253
```

Nonetheless, different executions were done by eliminating one of the preprocessing of the tokens such as not transforming to lowercase and by not removing the stopwords from the global dictionary.

By not removing the stopwords a MRR of 1.0 was obtained while by not transforming the tokens into lowercase an MRR = 0.8333 was also calculated.

5.1 What really happens

The resulting MRR obtained while not removing the stopwords is that in the retrieval with $qid = 3$ the right answer "The best mirror is an old friend" receives a better ranking than the scoring method using stopwords because the word "is" is found in the query and the answer. This must be really taken into consideration while evaluating the methods through metrics such as MRR and cosine similarity, as the results might be deceiving depending on the corpus on which the methods are being evaluated.

6 Discussion

The main goal was achieved as the AAE was completely programmed according to the requisites. This was achieved with ease as auxiliary classes were programmed which helped to handle the data to be processed. A brief comparison was made on what would happen if different parameters in the method were modified such as: not transforming the words into lowercase, not removing the stopwords or even by not removing the punctuation marks.

On the other hand, this project could go further by implementing a series of scorers that would calculate the Jaccard similarity or the edit distance to mention a few. By applying software engineering it would be desirable that those scorers are implemented in classes which derive from a general *Scorer* class. This class would have an abstract an *evaluate(Retrieval ret)* method which needs to be implemented by all the scorers. In this function all the answers would be scored according to the specific scoring method.

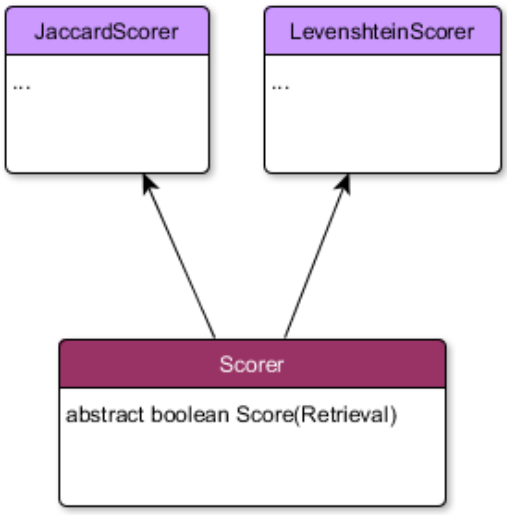

Figure 3: Scorer class

This implementation would also make easy to create a CPE with differente scoring annotators, where each annotator only needs to instantiate a different scorer object to create the comparisons. In that case, a minimal *Retrieval* annotation should be done and added into the index, in such a way that a final annotator would be able to make a comparison of all the results obtained from different scoring methods.