11-791 HW4 Report

Chen Sun <chen.sun@cs.cmu.edu>

1.Design Outline

The homework is generally provided with a formal framework to be implemented. Thus, we do not need to implement specific architecture here.

The general data flow of this project is as follows:

The ***Vector Space Retrieval*** controls the data to be input and the annotators to use in the program. Also, it parses the whole document lines and parse each line to be a document, as has been specified by the homework requirement. Finally, after all the processing, it prints the meta information of the project.

The ***Document Reader*** is the first step of processing. It reads out the queryId, relevance and put the document text in the Document object. This is a new data types defined in this homework. It converts a text line (or a document) to a Document object and stores them in the jcas indexes. However, the annotation here does not specify the range (beginning and ending) of the text, etc. This is because we no longer the annotation position here, since this information provides little relevance and each document is individually parsed.

After obtaining each document, now the ***Document Vector Annotator*** reading each document, and for each document, it parses the document's text part to tokens. The parsing details will be covered in later part of the report. After tokenizing the tokens, the tokens are stored in the document's token lists.

Obtaining the complete token lists, we now have each doc, queryId, relevance and tokens. Now, in ***Retrieval Evaluator***, we could have a full list to calculate the similarity based on token levels. For each tokens, convert them to HashMap of <String, Integer> terms, where the integer indicates the times the term appears in the token lists. Then, we could calculate the similarity of the tokens.

In this homework, I totally used three different kinds of evaluators to evaluate the similarity of two vectors, their results will be discussed in the following.

After all the evaluation of single queries and each documents. I firstly rank the documents according to their scores in descending orders. Secondly, read these documents lists, find the first relevant document and its rank in the list.

Finally, output the MRR of the document batches.

2. Implementation Details

In implementation details, I'd like to mention the parses, token choices and scoring I used.

1. For parsing the document, to improve the accuracies and mapping rate, several methods are used here.
   Firstly, set the delimiters to white space, comma, question mark and period.
   Secondly, convert all the words to lowercase, such as "Friends" to "friends".
   Thirdly, since we've been provided a stopwords list, I remove all the stop words from the queries so that the terms will be evaluated based on the unique terms.

I originally intend to use stemmer to trim the words to their original state, like “friends” to “friend”. However, due to lack of complete grammar rules, this may introduce more bias. Thus, I left the words untrimmed.
Before the converting cases, the rank of the fifth query was 2 and it turned to the first after trimming. Thus, the converting of cases may improve the accuracy and ability in finding relevant docs. These were done after the first several runs and initial error analysis.

2. For scoring part, each scoring method is an individual function taking the vectors of query and doc as its parameters. Thus, in testing the different scoring, only one line of code needs to be changed, in order to achieve to the low coupling and less work.
   For scoring methods, all three methods are adopted from wikipedia. The implementation were simple, just iterate the two sets and find the intersected part. Only slice details are different at the two levels.
3. Originally, I wish to use a HashMap to store all the docs that are of the same query ID and wish this could simplify the work. However, the UIMA tells internal exception and the token lists become null. So I can just give up the method and create new node storing the queryID, relevance and the vector and the scores. During this process, however, the original text is given up for easy processing.
4. For the type systems, I add an Query and Answer type, which both inherit from Doc. This is used to distinguish the difference between query and answer. Also, answer stores the score after the similarity analysis.
   However, these type systems were not used (they were commented out) in my code. This is simply because that the they were not able to be stored in the HashMap nor LinkedList, etc, making maintaining them extremely difficult.

3. Error Analysis

1. Cosine Similarity

If case is not converted, the result shows as belows.

```
Score:0.61237 rel=1 qid=1 sent1 rank=1
Score:0.15430 rel=1 qid=2 sent2 rank=1
Score:0.44721 rel=1 qid=3 sent3 rank=2
Score:0.15430 rel=1 qid=4 sent4 rank=3
Score:0.00000 rel=1 qid=5 sent5 rank=2
 (MRR) Mean Reciprocal Rank ::0.6666666666666667
Total time taken: 0.951
```

Especially for Query 5, that the relevant doc is extremely low and the score is zero. If I change to include the converting lowercase, the result is as follows.

```
Score:0.61237 rel=1 qid=1 sent1 rank=1
Score:0.46291 rel=1 qid=2 sent2 rank=1
```

Score:0.50000 rel=1 qid=3 sent3 rank=2
Score:0.18257 rel=1 qid=4 sent4 rank=1
Score:0.23570 rel=1 qid=5 sent5 rank=1
(MRR) Mean Reciprocal Rank ::0.9
Total time taken: 0.787

The running time drops because some words are combined, thus there are fewer terms in the vectors, decreasing the running time.
Secondly, all docs except the first have dramatic increase in scores. And the rank now is 1, 1, 2, 1, 1. All except the third doc is now the most relevant doc in the list.
To analyze the third document, we find that **"The best mirror is an old friend" and "One's best friend is oneself"** sentences. In this case, mirror refers to oneself. Also, the two sentences do not have similar meanings. In whichever case, if it cannot interpret the mirror and oneself in similar meaning, this relevant answer will always suffer from the lack of the semantic analyzing.
However, the MRR is 0.9, indicating that the evaluator shows a good evaluation process on token level. Thus, for the queries here, there is no need to use tf-idf to give different weights for different query terms.

For different methods, the evaluation do not show that significant effects.

2. Dice Coefficient

Score:0.50000 rel=1 qid=1 sent1 rank=2
Score:0.46154 rel=1 qid=2 sent2 rank=1
Score:0.50000 rel=1 qid=3 sent3 rank=2
Score:0.18182 rel=1 qid=4 sent4 rank=2
Score:0.22222 rel=1 qid=5 sent5 rank=1
(MRR) Mean Reciprocal Rank ::0.7
Total time taken: 0.799

Generally the ranks are lower that of the Cosine Similarity, since the intersected terms are counted once, but their times of appearances are ignored. Most docs, if they are relevant, the unique terms of both query and docs, tends to show more than one times, increasing weights on the docs.

3. Jaccard Coefficient

Score:0.33333 rel=1 qid=1 sent1 rank=2
Score:0.30000 rel=1 qid=2 sent2 rank=1
Score:0.33333 rel=1 qid=3 sent3 rank=2
Score:0.10000 rel=1 qid=4 sent4 rank=2
Score:0.12500 rel=1 qid=5 sent5 rank=1

(MRR) Mean Reciprocal Rank ::0.7
Total time taken: 0.867

It is quite similar to the Dice Coefficient, thus their results are similar, too. This method, together with the one listed above, only takes into consideration the unique terms. Thus, for short docs with only few words, the appearance of one term will be determined with a lot of weight. Thus, they are not appropriate for this kind of docs.
Also, each term in the Dice Coefficient and Jaccard Coefficient method, each term is regarded with equal weight, which goes against common sense. Thus, these two methods are not well posed here.