Report for Engineering and Error Analysis with UIMA

Chenying Hou
Andrew ID:chenyinh

October 28, 2013

1 Introduction

In this homework, I mainly complete three parts. First, the implementation of basic requirements and the results and error analysis. Second, based on the error analysis, I improve the retrieval system and improve the MRR performance. Third, I try other similarity measures such as dice coefficient, Jaccard coefficient and compare and analyse different similarity methods.

2 Basis Implementation of the System

2.1 Implementation Details

In this part, I have implemented all basis requirements of this task, which include automatically generating the type system implementation using UIMA >> JCasGen, correctly extracting bag of words feature vector from the input text collection, computing the cosine similarity between two sentences in the text collection and computing the MRR for all sentences in the text collection.

When extracting bag of words, I initially do not do any pre-processing for documents such as stop words removal and in next section I have implemented the pre-processing methods (including stop words removal, lower case transfer) and Section 2 will provide the results comparison for document pre-processing. In this part, I use **regular expression** in hw2 to find each token in a document. My system can tell and remove all useless punctuation for a term such as ".", "!", ";" and so on. Meanwhile it will maintain the punctuation "'" for words such as "one's".

I use a **HashMap** to generate the token list for each document and it can update its term frequency fast. I first put the token list into an *arraylist* and then transfer it using *Utils.fromCollectionToFSList* to *FSList*.

After computing the Cosine similarity, I sort the results according to the score of each document and compute MRR using that. In this part I present cosine similarity result and in Section 3 I present the different similarity scoring methods and their comparison.

2.2 Results and Error Analysis

The result for this part is as follows, we can see that MRR is *0.65*. After analysing the results, we found that the error produced because of the following reasons. First, for the words one with capital letter for first letter and one is not, my system thinks they are different terms, but actually they can be treated as the same. Second, for words such as *is*, *the*, *be* they are meaningless to some extend for a document, however my system considers them useful when generating tokens and computing similarity. Third, for some words such as *friend* and *friends*, actually they can be treated as the same, however, my system does not do that.

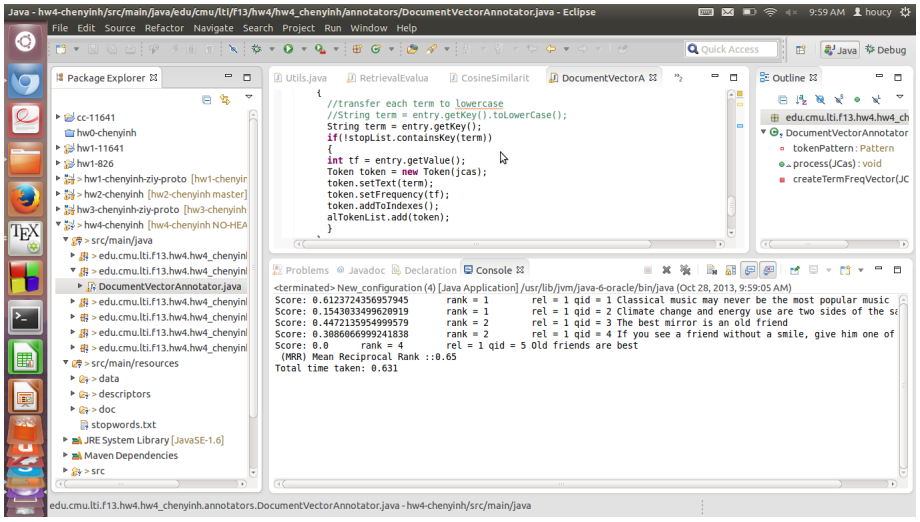

Figure 1: Initial Result for Cosine Similarity

3 Improved Implementation of the System

3.1 New Designs

Based on the error analysis of the previous section. I find two things matters the results. First is the preprocessing for generating tokens, it should be implemented to filter some useless words. Second when retrieving, not only the exact matching words should be considered, we should also consider prefix matching.

I put some new parts in my system and improve it by more accurately recognizing tokens and determining the similarity.

The changes are as follows.
First, for all terms read from the document, I first transfer them to lowercase format and then record them with token type.
Second, I use the stopwords list under the src/main/resource dictionary and when generating tokens I ignore the stop word in the document.(Using HashMap to implement the stopword match).
Third, I also try the prefix matching and consider for example *friend* and *friends* as the same.

3.2 Result and Analysis

The experiment result is shown in Figure 2. We can see that the MRR improved to 0.8 and for detailed comparison we found that the improvement achieved exactly because of the new strategies we adopt. For example, for the document "Old friends are best", now my system can correctly rank it to the first place. However, due to the size and diversity of the test documents, the stopwords removal effect is not significant in this experiment. I believe it will work well in true practise with large number of queries and large size of documents.

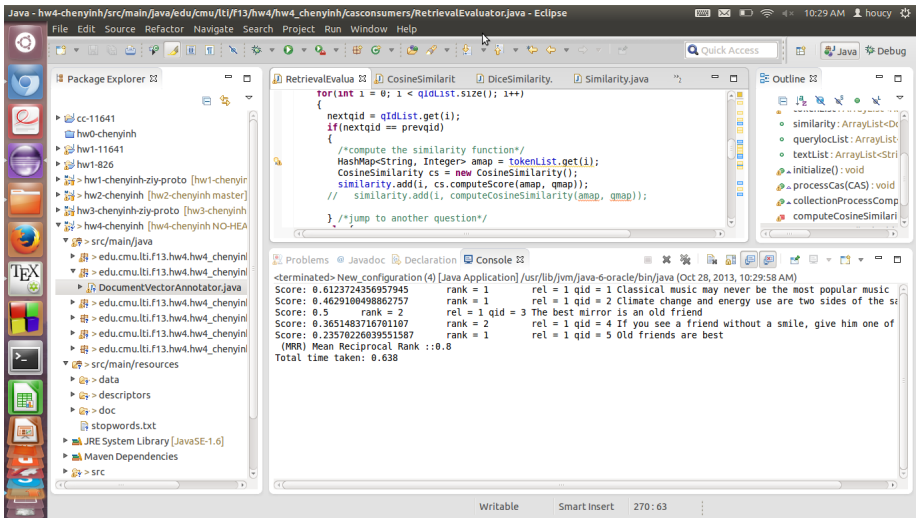

Figure 2: Improved Result for Cosine Similarity

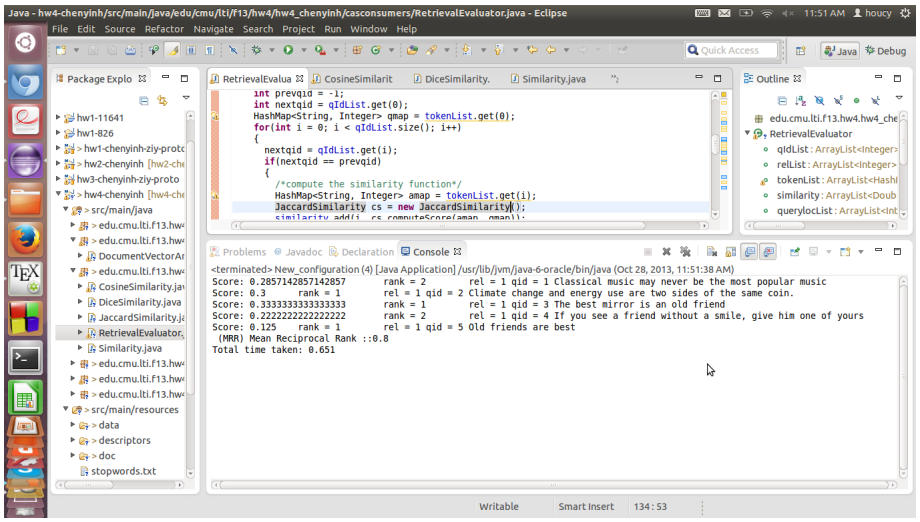

Figure 3: Result for Jaccard Similarity

4 Other Similarity Measurement Comparison

I also tried two other similarity functions besides Cosine similarity. One is Dice coefficient and the other is Jaccard coefficient.

For Vector A and Vector B, Cosine similarity is defined as $(\sum_{i=1}^{|V|} a_i * b_i)/\sqrt{\sum_{i=1}^{|V|} a_i^2} * \sqrt{\sum_{i=1}^{|V|} b_i^2}$.

Jaccard coefficient is defined as the size of the intersection divided by the size of the union of the query vector and document vector. Assume that A represents query vector and B represents the document vector. The formula is $J(A,B) = |A \bigcap B|/|A \bigcup B|$, and $A \bigcap B$ means the tokens existing both in A vector and B vector and $A \bigcap B$ represents the tokens existing in A vector or B vector.

Dice coefficient is similar to Jaccard coefficient. The formula of it is $D(A,B) = 2 * |A \bigcap B|/(|A| + |B|)$. we can see that compared with Jaccard coefficient, Dice distance cares more about the overlap part of A and B. Dice distance is more sensitive in heterogeneous data sets and it gives less focus on the outliers.

From the above formulas, we can see that Cosine similarity penalizes less in different amount of non-zero entries compared with Jaccard coefficient and Dice coefficient. And that is a very good attribute in language processing since different documents may has different size but we cannot determine that they are unrelated with each other. And for Jaccard coefficient and Dice coefficient we

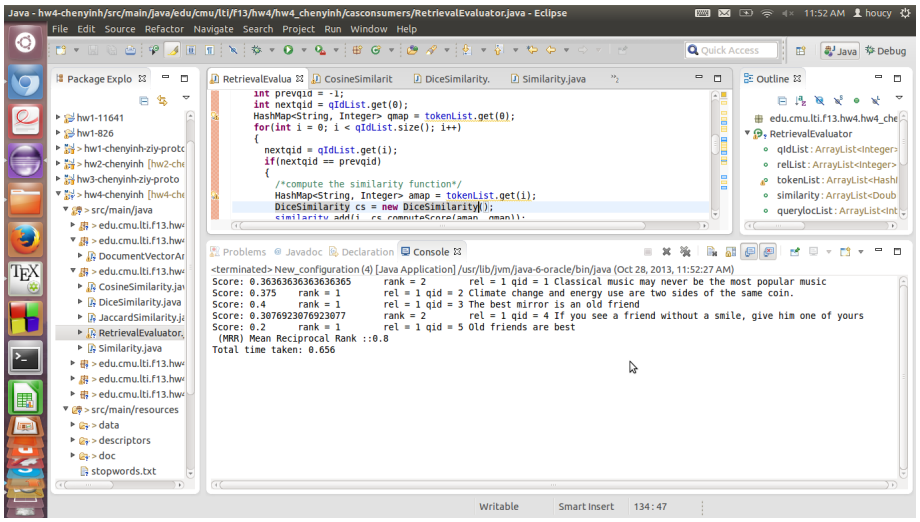

Figure 4: Result for Dice Similarity

can see that Jaccard gives lower similarity score for low-overlap sets(two vectors with small overlap).

The experiment results are showed in Figures 3, 4 below (cosine similarity in Figure 2).

From the results we can see that the difference between Dice similarity and Jaccard similarity's difference is smaller compared with Cosine similarity and that is consistence with their formula's difference. They perform differently on different documents. In this small data set, the MRR value are the same under three different methods, but when in more diverse and large data set, I think Cosine similarity will perform better based on the above analysis.

5 System Design and Algorithmic Design

In my system, I use the Template design pattern when calculating similarities. Since I implement three different similarity functions. Their procedures are more or less similar. So I construct a abstract class called *Similarity*, I make two abstract methods called *computeScore* and *length* since different similarity functions have different way to evaluate length and score. Then I build a general method (realized) called *countUninion* to calculate the intersection between two vectors. Then each specific similarity class such as *CosineSimilarity*, *JaccardSimilarity* or *DiceSimilarity* is all extended from *Similarity* and realizes the abstract method specifically.

In order to speed up the running time of my system. I use *HashMap* a lot (both in stopwords removal and tokenlist generating). I also use *collection.sort* method to sort the document according to similarity score.