11791 Homework 3 Report

Fei Xia
Language Technology Institute
feixia@andrew.cmu.edu

In this report, we will first briefly talk about the Task 1, i.e. building vector space retrieval model using UIMA. Then we will focus our main attention on Task 2 – do error analysis and implement new strategies to improve the performance.

1 Task 1

As the framework has been provided, we only need to finish the annotator and the consumer. Following the instruction, in the annotator, I used the simple white-space tokenization algorithm to populate the token list of the corresponding doc. In the consumer, I constructed some data structures in the "processCas" method. Then in "collectionProcessComplete" method, I used these data structures to compute the similarity and the Mean Reciprocal Rank. To make the code clean, I also created a new class called "DocVec" to represent a doc. It contains all the information we need to do the vector space retrieval and to output the final results, such as the vector representation of the doc, the similarity, the rank, etc.

The MRR of task 1 is 0.4375.

2 Task 2

2.1 Error Analysis

MRR=0.4375 is definitely not very good in this small dataset, so we try to do some error analysis. In my setting, if the relevant document is not ranked in the first place, there we call it an error. By inspecting the final output, I found that all queries except the one with qid=10 have errors and I classify these errors to four different types. It is worth to note that there may be multiple reasons for a retrieval system to make an error, thus one query error may be classified into multiple types in my setting and the sum of the percentage may not be equal to 100%. Please refer to Table 1 for the statistics.

Table 1: The Error Type Statistics

Error Type	qid	percentage
Tokenizaton	2, 5, 6, 20	21.05%
Stop Words	3, 14, 16	15.79%
Stemming	3, 6, 16	15.79%
Similary Measure	1, 4, 7, 8, 9, 12, 13, 15, 18, 19	52.63%

Let's look at the four types of errors one by one.

- Tokenization. Not using a good tokenization algorithm is one main reason of the error. For example, the query 2 is:

  What has been the largest crowd to ever come see Michael Jordan

  And the relevant document is:

  When Michael Jordan–one of the greatest basketball player of all time–made what was expected to be his last trip to play in Atlanta last March, an NBA record 62,046 fans turned out to see him and the Bulls.

  Because of the simple white space tokenization, the key word "Jordan" in query won't be matched with "Jordan–one" in the document.

- Stop Words. Not filtering out the stop words is one main reason of the error. For example, the query 3 is:

  In which year did a purchase of Alaska happen?

  And the relevant document is:

  Alaska was purchased from Russia in year 1867.

  There are lots of stops words in these sentences, such as "in", "of", "a", "from", etc. Eliminating the stop words would make the two sentences' similarity higher.

- Stemming. Not using a stemming algorithm is one main reason of the error. For example, the query 6 is:

  Who was the first person to run the mile in less than four minutes

  And the relevant document is:

  Roger Bannister was the first to break the four-minute mile barrier.

  The "minutes" and "minute" should match, but because there is no stemming process, they cannot be matched.

- Similarity measure. Not using a good similarity measure is one mean reason of the error. I think that this is the most common error type. Simply represent a document with its word frequency is not enough, we can consider to use TF-IDF features.

2.2 Improvement

Based on the error analysis in Section 2.1, I have designed and implemented a better system – including better tokenization, stemming, stop words filtering and other similarity measures. Let's talk about their effects respectively, then combine all these components and see what the performance would be.

2.2.1 Inspect Each Component Respectively

So we have four components – better tokenization, stemming, stop words filtering and better similarity measure. I use LingPipe to implement these components. We consider that the better tokenization as a basic component, so we will compare the performance based on that. Table 2 gives a summary, where WST means White Space Tokenizer and LPT means LingPipe Tokenizer.

Table 2: Inspect Each Component Respectively

WST	LPT	LPT + Stop Words Filter	LPT + Stemmer	LPT+ TF-IDF
0.4375	0.5250	0.5417	0.5333	0.5083

- White Space Tokenizer (WST) V.S. LingPipe Tokenizer (LPT). The MRR is increased to 0.5250. Going back to see the query 2 example in Tokenization error:

  What has been the largest crowd to ever come see Michael Jordan

  We found that the similarity of the relevant document is increased from 0.2858 to 0.3032 and the rank becomes 1, which is 2 originally.

- LP Tokenizer only V.S. LP Tokenizer + Stop Words Filter. I built a stop words set based on the given *stopwords.txt* in the archetype. The MRR is increased to 0.5417. Going back to see the query 3 example in Stop Words error:

  In which year did a purchase of Alaska happen?

  We found that the similarity of the relevant document is increased from 0.2108 to 0.4000 and the rank becomes 1, which is 2 originally.

- LP Tokenizer only V.S. LP Tokenizer + Porter Stemmer. The MRR is increased to 0.5333. For the query 6 example in Stemming error:

  Who was the first person to run the mile in less than four minutes

  we found that the similarity of the relevant document is increased from 0.5625 to 0.625, and the rank becomes 1.

- LP Tokenizer only V.S. LP Tokenizer + TFIDF. Interestingly, we found that the MRR is decreased to 0.5083. Does that mean TFIDF is not a good similarity measure feature? Here I claim that TFIDF can work pretty well but should be in a situation that the documents have been cleaned – i.e. after filtering out stop words, stemming etc. We will later see some demonstration on this.

2.2.2 Combine all Components

In this section, we gradually combine all components and then focus on changing the similarity measure. Starting from the vector space retrieval model in task 1, We list our steps and the MRR at that point:

1. Step 1: Add LingPipe Tokenizer, the MRR is 0.5250.
2. Step 2: Add Stop Words Filter, the MRR is 0.5417.
3. Step 3: Add converting letters to lower cases, the MRR is 0.5750.
4. Step 4: Add porter Stemmer, the MRR is 0.6292.
5. Step 5: Change the similarity measure to Edit Distance, the MRR is 0.5292.
6. Step 6: Change the similarity measure to Jaccard coefficient, the MRR is 0.6542.
7. Step 7: Change the similarity measure to TFIDF, the MRR is 0.7292.

So after the document pre-processing – tokenization, stop words filtering, converting letters to lower cases, stemming – we can see in this dataset, the performance rank of similarity measure is: TFIDF > Jaccard coefficient > vector space model > edit distance. TFIDF works the best because it not only considers the word frequency, but also the word importance. Higher word frequency doesn't necessarily mean the word should be given more weight. We have to consider whether the word is common in the whole collection. TFIDF works well also demonstrates my claim in the previous section.

2.2.3 System Design

As I have implemented three other similarity measures, it is important to design a system that can reuse the codes for each similarity measure. There is no fancy design in my system, but it is clean and concise – when we want to change the similarity measure, we can simply plug in that similarity function and everything else remains the same. The idea is that I created a new class that is called GeneralDocRept, which is a general document representation class. That class contains everything we need in doing the retrieval, so each similarity function would directly modify the similarity value stored in this class and all other stuff like sorting, constructing output, etc can remain the same.

Following the instruction, the code of task 2 won't be submitted.

3 Others

My program can run very well in my computer. If you cannot run it, please contact me via feixia@cs.cmu.edu. Thank you!