HW3-Report

Sun Xing, andrew ID: xings

October 22, 2014

Table of contents

1 Introduction

In this project, we built a system to evaluate whether the retrieved documents contains the information needed to answer a particular query. In task1, a simple system was implemented using simple tokenization algorithm and cosine similarity to measure the similarity between the query and the retrieved document and determine the rank of the retrieved document accordingly. No stemming was used. The simple system achieved 0.4375 MRR on the testing data, which is disappointing since this score is comparable to randomly rank all the retrieved document. In this report we analyze the errors produced by this system, propose ways to improve the system and evaluate the improvement achieved.

2 Error Analysis

There are all sorts of error occurring in the simple project. This section lists some of the obvious errors in the test result.

2.1 Effect of length of documents

The cosine similarity is biased towards a certain length of document. In the simple case, form the formula of cosine similarity, we know that with the same frequency of word appeared in both query and document, the longer the document, the lower the cosine similarity. This can be reflected in the set of document in qID =1. The shortest one has the higher similarity score (though there are other contributing factor)

2.2 Forms of words

The words of different form are considered as different words in our system in task 1. (e.g. egg and eggs are considered different words) Obviously, this will cause significant mis-calculation of the similarity between the query and document.

2.3 Tokenization problems

The tokenization system fails of consider punctuations and does not lowercase in our system in task 1. A simple example of the problem caused by this is that all the word followed by any punctuations cannot be recognized correctly.

2.4 Incapable of recognizing expression with similar semantic meaning

The current system is not capable to recognize words with similar semantic meaning. One example is in qID=9 “spacecraft” and “spaceship” are of the same meaning but has not contribution to the similarity between the query and the retrieved document.

2.6 Misspelling

In the document given, there are several words misspelled. (e.g. Pompei in line 3, qID=1) Our system in task 1 cannot automatically correct the spelling and directly recognize them as different words.

2.7 What/When/Where/Which problem

There are questions that start with what/when/where, etc. And they requires the answer contains certain named entity e.g. name of person/year/name of place. The current system does not deal with this problem. An example would be in qID=3 where the question contains "when".

It is not easy to assign the incorrect retrieval of document to a certain type of error. Here listed the number of query that may be affected by a certain error.

Error Type	Frequency
Length of Documents Bias	20
Forms of Words	11
Tokenization Problem	16
Words with Similar Semantic Meaning	7
Misspelling	4
What/When/Where/Which problem	15

3 System Design

3.1 architectural aspect

One interesting type of error observed is the What/When/Where/Which problem. Here I propose a way to solve the problem by adding an extra layer to the current system architecture. Before evaluation of the documents, we first revisit the query to see if there is a specific named entity that the query is looking for. For example, question starts with "what year" is looking for a named entity of a year, question starts with "who" is looking for a named entity of a person's name. Then we uses a NER to detect if the retrieved document has the according named entity. If it has, we can pass the document to the evaluator, otherwise we can directly ignore this retrieved document.

3.2 algorithmic aspect

3.2.1 tokenizing algorithms

In the error analysis, we can see that problems caused in the tokenization step can be categorized in to 1) fails to consider punctuation and 2) fails to deal with uppercase and lowercase words. Therefor we can modify the tokenization algorithm so that they can ignore the punctuation.

To further improve the system, noticed that the length bias is a practical concern, we can try to alleviate this problem by ignoring some of the common words such as in/at/is/or, etc. This can reduce the irrelevant words in the document thus they do not contribute to the similarity matrix. A list of stop words is stored in /resources/stopwords.txt

Another way to improve the tokenization algorithm is to directly employ the coreNLP tokenizer. It can deal with more complicated cases.

3.2.2 stemming algorithms

In the error analysis, the "form of words" problem can be addressed by using a stemming algorithm. Here I propose to use the **Stanford Lemmatizer** to do the stemming. It converts the words back to its original form so that it is easier to compute the similarity between the query and the retrieved document.

3.2.3 similarity measures

The cosine similarity is not a very good way to measure the similarity between the query and the retrieved document due to many reasons: it does not consider the order of words (it is a **bag of words model**), it is biased toward shorter documents, etc. The good news is that we have a pool of other existing similarity measures here we introduce some of them.

TF-IDF: TF-IDF treats each document as a vector. IDF brings document frequency into consideration thus address the issue that rare terms contains more information than frequent terms so they need to be weighted differently. Instead of calculating Euclidean distance it calculates the angle between two document vectors and use it as the similarity measures. This makes it capable of take the order of words into consideration

Jaccard coefficient: Jaccard coefficient merely measures the overlap between two corpses. In Jaccard coefficient, the term frequency and the document frequency is not considered.

Dice coefficient: Similar to Jaccard coefficient, it measures the overlap between two corpses and it does not consider frequencies and order. Jaccard coefficient and Dice coefficient are similar to cosine similarity except they do not consider frequencies.

4 Improving Performance

4.1 Improvements

I wrote a tokenizer based on the ideas written in 3.2.1. The Stanford Lemmatizer, which will be discussed in 4.2 deals with the uppercase/lowercase issue so we do not do the conversion in the tokenizer. I tried to use Stanford Lemmatizer in document vector annotator.

I also explored some of the similarity measures, this is written in part 5.

4.2 Results Evaluation

In this part the tokenizer and Stanford Lemmatizer was implemented and the corresponding MRR was calculated. (similarity measures are discussed further in the next part) The results are listed below.

MRR value	Not use Stanford Lemmatizer	Use Stanford Lemmatizer
original tokenizer	0.4375	0.5500
improved tokenization algorithm	0.4708	0.5833

As can be seen, both the improved tokenization algorithm and the Stanford Lemmatizer improved the MRR value. The combination of both result in a MMR of 0.5833, +0.1458 more than the original system.

5 Extra Credit

5.1 Experiment

In the bonus credit part we will explore the effect of using different similarity measures by playing with Dice coefficient and Jaccard coefficient. The two similarity measures' performance are evaluated on different setup. (use/not use Stanford Lemmatize, use original tokenizer/improved tokenizer)

5.2 Result

The result of my experiment are shown in the tables below.

Jaccard coefficient

MRR value	Not use Stanford Lemmatizer	Use Stanford Lemmatizer
original tokenizer	0.4500	0.5000
improved tokenization algorithm	0.4875	0.5333

Dice coefficient

MRR value	Not use Stanford Lemmatizer	Use Stanford Lemmatizer
original tokenizer	0.4500	0.5000
improved tokenization algorithm	0.4875	0.5333

5.3 Analysis

On the given samples, the two similarity measures obtained the same result. Though the similarity scores are different, they produce similar rankings thus result in the same MRR scores in all of the setups. This can be explained by looking into the two coefficients' formula

Dice Coefficient

$$QS = \frac{2C}{A+B} = \frac{2|A \cap B|}{|A| + |B|}$$

Jaccard Coefficient

$$J(A,B) = \frac{|A \cap B|}{|A \cup B|}.$$

If we divide Dice coefficient by 2, it dose not affect the rankings, and then the difference between the two coefficients are on the denominator. The denominator of Jaccard coefficient is the denominator of Dice coefficient minus the number of overlapping terms between the query and the retrieved document. If we look into the sample document, we can see that the number of overlapping terms with the query are similar among the retrieved documents. This explains the same MRRs of the two different similarity measures.

5.4 Checking for Statistical Significance

Question remains whether one of the three similarity measures are better than the other (in a given system). Since Jaccard coefficient produces the same result to the Dice coefficient. Here we only compare the performance of cosine similarity and Dice coefficient under the system that uses the improved tokenizer and Stanford Lemmatizer. Here we use t test to derive the p value. First we randomly divide the sets of query/documents in document.txt into 5 groups, each contains 4 sets, then calculate the t score accordingly.

From the calculation I obtained t = 0.5017, the two tailed P value is 0.6504.

The P value is quite large so we cannot conclude statistical significance. The difference between the two similarity measures are not big enough and the dataset is not big enough to conclude statistical significance.