Yiran Fei
yiranf
October 20, 2014

HW3-Report

Software Method

Task 1

(Branch: Master/Released)

Requirements

1. Implement the collection reader using a simple white-space tokenization algorithm, which does not split on punctuation and considers a sequence of adjacent white-spaces to be a single separator. It also does not lowercase.
2. Design and build a Vector space Retrieval Model based on UIMA framework.

Design

1. Collection Reader

In Task 1, we do not need to consider the punctuations, upper/lower case issues, so just use a simple splitter which can divide the sentence into many tokens by one or more whitespace. Using the provided Type System, the tokens are stored as a FSList and passed to the AEs.

2. Vector space Retrieval Model

The AE part of this system is actually a score calculator and a ranking system. For each sentence among a set of statements belong to a query, the calculator needs to compare it with the query and get a cosine similarity value. Then, the ranking system should rank sentences and output the pre-determined "right" result and its rank in this set in order to evaluate the performance of this algorithm.

Although there's no need to sort all the sentences to get the rank of a specific output sentence, I do perform a sort among each query set in order to get a full report for the error analysis in the Task 2.

In the end, a MRR which can represent the total performance should be outputted to the report.

Algorithm (From Wiki)

1. **Cosine Similarity**

$$\text{similarity} = \cos(\theta) = \frac{A \cdot B}{\|A\|\|B\|} = \frac{\sum_{i=1}^{n} A_i \times B_i}{\sqrt{\sum_{i=1}^{n} (A_i)^2} \times \sqrt{\sum_{i=1}^{n} (B_i)^2}}$$

2. **MRR**

$$\text{MRR} = \frac{1}{|Q|} \sum_{i=1}^{|Q|} \frac{1}{\text{rank}_i}.$$

Test Result

Using the whole provided dataset, the MMR of the system in Task 1 is 0.4375.

Task 2

(Branch: Task2/Not Released)

Error Analysis

I modified the output file by adding more information in order to do error analysis. Here is an example of Query 1-3 (The whole file can be found on github):

```
cosine=1.0000   rank=0  qid=1   rel=99  Give us the name of the volcano that destroyed the ancient city of Pompeii
cosine=0.6396   rank=1  qid=1   rel=0   Vesuvius is located near the ruins of the destroyed city of Pompeii.
TokenName           NoInQ     NoInS
of                  2         2
destroyed           1         1
the                 3         2
city                1         1
cosine=0.2791   rank=2  qid=1   rel=1   In A.D. 79, long-dormant Mount Vesuvius erupted, burying in volcanic ash the Roman city of Pompeii; an  estimated 20,000 people died.
TokenName           NoInQ     NoInS
of                  2         1
the                 3         1
city                1         1
cosine=0.2446   rank=3  qid=1   rel=0   You can see Vesuvius in the background, near ruins of Pompeii; its last eruption was in 1944.
TokenName           NoInQ     NoInS
of                  2         1
the                 3         1
cosine=0.2068   rank=4  qid=1   rel=0   In 79 A.D., this ancient city was buried in an avalanche of hot ash from Mount Vesuvius.
TokenName           NoInQ     NoInS
of                  2         1
ancient             1         1
city                1         1


cosine=1.0000   rank=0  qid=2   rel=99  What has been the largest crowd to ever come see Michael Jordan
cosine=0.3018   rank=1  qid=2   rel=0   A supposedly last play of  Michael Jordan gathered some of the largest crowd in history of NBA.
TokenName           NoInQ     NoInS
Jordan              1         1
the                 1         1
crowd               1         1
Michael             1         1
largest             1         1
cosine=0.2858   rank=2  qid=2   rel=1   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.
TokenName           NoInQ     NoInS
to                  1         3
see                 1         1
the                 1         2
Michael             1         1
cosine=0.1987   rank=3  qid=2   rel=0   In the Bulls' last visit to Atlanta, an NBA-record 62,046 fans showed up at the Georgia Dome.
TokenName           NoInQ     NoInS
to                  1         1
the                 1         2
cosine=0.1952   rank=4  qid=2   rel=0   The Immortal World Tour of Michael Jackson gathered some of the largest crowds of fans in the history of pop music.
TokenName           NoInQ     NoInS
the                 1         2
Michael             1         1
largest             1         1


cosine=1.0000   rank=0  qid=3   rel=99  In which year did a purchase of Alaska happen?
cosine=0.4216   rank=1  qid=3   rel=0   William Seward negotiated a purchase of Alaska for $7.2 million.
TokenName           NoInQ     NoInS
of                  1         1
a                   1         1
purchase            1         1
Alaska              1         1
cosine=0.2673   rank=2  qid=3   rel=0   1867 - U.S. President Andrew Jackson proclaims treaty for purchase of Alaska from Russia.
TokenName           NoInQ     NoInS
of                  1         1
purchase            1         1
Alaska              1         1
cosine=0.2357   rank=3  qid=3   rel=1   Alaska was purchased from Russia in year 1867.
TokenName           NoInQ     NoInS
year                1         1
Alaska              1         1
```

According to the whole report, I classify the errors as the following tab:

No.	Type	Description	Example (QId)
1	Stopwords	A lot of meaningless stop word affects the evaluation	1, 3, 5, 14
2	Mismatch	1) Upper/Lower Case	9,
		2) Voice (Active/Passitive)	5,
		3) Tense	N/A
		4) Part of speech	13,
3	Redundancy	Too many tokens in the sentence will reduce the cosine value	2, 15, 20
4	Order	Sometimes the order of tokens is important to the meaning of a sentence	N/A
5	Synonym	Some words are different but have similar meaning	1,
6	Numeric Value	When the answer is a numeric values, it's always confusion (Time, money).	3, 4, 7, 8
7	Repeated Query	Some sentences are another form of the query, and they always get a high scores.	17,
9	Short Answer	The answer which is extremely short can get a low score.	11,
10	Punctuation	The tokenization function is too simple	1, 2, 5. 19

Design

In order to be more flexible while comparing different algorithms and token/stem functions, I do not design different analysis engines for each algorithm. I just add these algorithms as functions in the retrieval evaluator class. And during the test, I can easily execute different algorithms by calling

Improvement

1. Tokenization

Use a tokenizer provided by the Apache Lucene (This is the only place I use the Lucene, other algorithms are written by myself). This tokenizer will divide every token from a sentence by (multiple) space, punctuations and other common dividers. After adding this tokenizer to the cosine similarity algorithm, the MRR raises from 0.43 to 0.53. See codes and report in commit.

2. Stemming

According to the Error Analysis, many words have different formats. Using a stemming function can reduce this kind of errors. After applying a stemming function, the MRR can be improved from 0.53 (Based on the version adding tokenizer) to 0.54. See codes and report the commit.

3. Similarity Calculation

I implemented 5 different algorithms to calculate the similarity values and make a comparison between them.

3.1 Jaccard index

The Jaccard index has two values whose sum are 1. In this implementation I use Jaccard similarity instead of Jaccard distance:

$$J = \frac{M_{11}}{M_{01} + M_{10} + M_{11}}.$$

This algorithm ignores the frequency of each token in the sentence. The test result is a little worse than cosine similarity, which drops from 0.53 to 0.46. See Codes and report in commit.

3.2 Dice coefficient

The Dice algorithm compute the similarity score by counting the common elements between the query and the document.

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

The Dice algorithm also ignores the frequency of each token. However, although it gets different scores, the performance is just the same with the Jaccard algorithm (0.46). See codes and report in commit.

3.3 Tversky index

The Tversky index is kind of the combination of Dice and Jaccard with 2 parameters which can be modified to improve the performance.

$$S(X,Y) = \frac{|X \cap Y|}{|X \cap Y| + \alpha|X - Y| + \beta|Y - X|},$$

The result is quiet different when we use different alpha and beta value:

	alpha	beta	MRR
1	0.1	0.9	0.5292
2	0.2	0.8	0.5292
3	0.3	0.7	0.4958
4	0.4	0.8	0.4917
5	0.5	0.5	0.4667
6	0.6	0.8	0.5167
7	0.7	0.3	0.5167
8	0.8	0.8	0.5083
9	0.9	0.1	0.5167

Some result are dramatically the same, I think this mainly due to the small size of the test data. See commit.

3.4 Okapi BM25

BM25 is a popular algorithm in search engine. It based on the idf which can evaluate the importance of a token in the document:

$$\mathrm{idf}(t, D) = \log \frac{N}{|\{d \in D : t \in d\}|}$$

And the score of BM25 can be calculated by:

$$\mathrm{score}(D, Q) = \sum_{i=1}^{n} \mathrm{IDF}(q_i) \cdot \frac{f(q_i, D) \cdot (k_1 + 1)}{f(q_i, D) + k_1 \cdot (1 - b + b \cdot \frac{|D|}{\mathrm{avgdl}})},$$

With k_1 = 1.2, b = 0.75, the MRR of result is 0.7533. It's the best result among all the test. See codes and report commit.

4. Deleting Stop words

In my opinion, deleting the stop words are quiet different from other tokenizing/stemming method. Because the cosine similarity is related to the frequency of words while Dice/Jaccad/Tversky ignores the term frequency. However, in most cases, the frequency of some stop words are larger than other meaningful words. See commit.

In order to get a better view of the impact of deleting stop words, I do this test at last.

	With Stopwords	Without Stopwords	Diff
Cosine	0.5458	0.5083	-0.0375
Jaccad	0.4667	0.5125	0.0458
Dice	0.4667	0.5125	0.0458
Tversky (alpha = 0.2)	0.5292	0.5542	0.025
BM25 (k_1 = 1.2, b =0.75)	0.7533	0.6058	-0.1475

According to the test, it's obvious that deleting these stop words reduces the accuracy of the cosine algorithm and BM25, while increase the accuracy of other frequency-ignorance algorithms.

Conclusion

1. The BM25 are the most efficient algorithm in this test. It considers a large number of attributes of the whole text collections and the single document, which must be an important reason for its high MRR.
2. During the error analysis, I thought that many high-frequency stop words may be an important cause of some error. However, after the comparison of the different performances, I found that these stop words are useful to some frequency related algorithm, while harmful for some algorithms that ignores the term frequency.