Dishan Gupta

dishang@andrew.cmu.edu

ASSIGNMENT 4

1 Task 1

The purpose of task 1 was to design a simple vector space retrieval system using the UIMA framework. In particular, the cosine similarity, Jaccard index and Dice coefficient measures were used as scoring metrics and based on the scores I ranked the retrieved documents for each query. Subsequently, I computed the Mean Reciprocal Rank (MRR) for the system for each scoring measure. MRR is a useful metric when your system is supposed to rank the correct document at the first position. The mean reciprocal rank is an average of the reciprocal ranks of results for a sample of queries Q:

$$MRR = \frac{1}{|Q|}\sum_{i=1}^{|Q|}\frac{1}{rank_i}$$

1.1 Type System

The type system contains the Document and Token data types. Described below are some of their aspects:

- Document - The Document data type contains the following information as features: Relevance Value, Query ID, Text String, Token List. The Token List is basically an FS List representing 'bag-of-words' Token types for each document.
- Token - The Token data types contains the token text string and its corresponding frequency (in this case number of occurrences in the document) as features.

These data types and their features are sufficient to build efficient annotators for the system.

1.2 Analysis Engine

The aggregate analysis engine is composed of three primitive analysis (described in the same order as present in Fixed Flow):

- DocumentReader - Document Reader extracts the relevant value, query id number, and sentence text from the text collection. This information is kept in CAS for further processing. Document Reader descriptor is available at descriptors/retrievalsystem/DocumentReader.xml. The corresponding Java implementation is available at edu.cmu.lti.f13.hw4.hw4_dishang.collectionreaders.DocumentReader.java.
- DocumentVectorAnnotator - DocumentVectorAnnotator is an annotator which processes the Document annotations made in Document Reader and populates the TokenList feature corresponding to that annotation. To construct tokens it uses the Stanford CoreNLP pipeline imported from the Maven repository. It stores the tokens and their corresponding frequency in a HashMap and then iterates over it to construct an ArrayList of Token types. Finally, it converts

the ArrayList to an FSList using a utility method Utils.fromCollectionToFSList and stores it into the Document data type feature TokenList for that particular Document annotation. Its descriptor is available at descriptors/retrievalsystem/DocumentReader.xml. The corresponding Java implementation is available at edu.cmu.lti.f13.hw4.hw4_dishang.annotators.DocumentVectorAnnotator.java.

- RetrievalEvaluator - RetrievalEvaluator is a CAS consumer class which computes the cosine similarity, Jaccard Index and Dice coefficient measures for the query and the document and ranks the documents accordingly. Subsequently, it calculates the MRR for the retrieval system based on each scoring metric separately. Its descriptor is available at descriptors/retrievalsystem/RetrievalEvaluator.xml. The corresponding Java implementation is available at edu.cmu.lti.f13.hw4.hw4_dishang.annotators.RetrievalEvaluator.java.

The aggregate analysis engine essentially executes the functionality of each of its primitive analysis engines in the order described above. The aggregate analysis engine descriptor is available at descriptors/retrievalsystem/VectorSpaceRetrieval.xml. The corresponding Java implementation is available at edu.cmu.lti.f13.hw4.hw4_dishang.VectorSpaceRetrieval.java.

1.3 Results

Three scoring metrics were used to compare the similarity between a query and its retrieved documents. The results are shown below:

1.3.1 Cosine Similarity

Given two vectors of attributes, *A* and *B*, the cosine similarity, *cos(θ)*, is represented using a dot product and magnitude as:

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

The console output for simple cosine similarity based MRR is shown below:

```
Score: 0.4522   rel=1   rank=1  qid=1   Classical music may never be the most popular music
Score: 0.0980   rel=1   rank=1  qid=2   Climate change and energy use are two sides of the same coin.
Score: 0.4629   rel=1   rank=1  qid=3   The best mirror is an old friend
Score: 0.2500   rel=1   rank=3  qid=4    If you see a friend without a smile, give him one of yours
Score: 0.0000   rel=1   rank=2  qid=5    Old friends are best

 (MRR) Mean Reciprocal Rank ::0.7666666666666667
```

From the above console output, it can be seen that the cosine similarity using simple token matching does not do well for the 4th and 5th query. Thus, we construct document vectors with lower case tokens and the results are shown below:

Score: 0.4522 rel=1 rank=1 qid=1 Classical music may never be the most popular music
Score: 0.2941 rel=1 rank=1 qid=2 Climate change and energy use are two sides of the same coin.
Score: 0.4629 rel=1 rank=2 qid=3 The best mirror is an old friend
Score: 0.2500 rel=1 rank=3 qid=4 If you see a friend without a smile, give him one of yours
Score: 0.1581 rel=1 rank=1 qid=5 Old friends are best

(MRR) Mean Reciprocal Rank ::0.7666666666666667

The above output indicates that lower case token vectors have the potential to improve the system's retrieval performance as it is able to do well on the 5th query. But, it degrades on the 3rd query (the relevant document is now ranked 2). This suggests that using a weighted combination of scores for normal and lower case tokens should perform well. Mathematically,

$$Score = w * cosSim(queryN, docN) + (1 - w) * cosSim(queryL, docL)$$

where, queryN, docN are the normal token vectors, queryL, docL are the lower case token vectors and w is the weight.

After experimenting with w, w=0.4 produced the best results:

Score: 0.4522 rel=1 rank=1 qid=1 Classical music may never be the most popular music
Score: 0.2157 rel=1 rank=1 qid=2 Climate change and energy use are two sides of the same coin.
Score: 0.4629 rel=1 rank=1 qid=3 The best mirror is an old friend
Score: 0.2500 rel=1 rank=3 qid=4 If you see a friend without a smile, give him one of yours
Score: 0.0948 rel=1 rank=1 qid=5 Old friends are best

(MRR) Mean Reciprocal Rank ::0.8666666666666668

All the queries, except the 4th, retrieved a relevant document at the first position.

1.3.2 Jaccard Index

The Jaccard coefficient measures similarity between sample sets, and is defined as the size of the intersection divided by the size of the union of the sample sets:

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

As previously described, a weighted combination of normal and lower cased tokens vectors was used to compute scores using Jaccard index. After experimenting with w, w=0.2 produced the best results:

Score: 0.2000 rel=1 rank=1 qid=1 Classical music may never be the most popular music
Score: 0.1433 rel=1 rank=1 qid=2 Climate change and energy use are two sides of the same coin.
Score: 0.3000 rel=1 rank=1 qid=3 The best mirror is an old friend
Score: 0.1000 rel=1 rank=3 qid=4 If you see a friend without a smile, give him one of yours
Score: 0.0615 rel=1 rank=1 qid=5 Old friends are best

(MRR) Mean Reciprocal Rank ::0.8666666666666668

1.3.3 Dice Coefficient

The Dice coefficient can be viewed as a similarity measure over sets:

$$s = \frac{2|X \cap Y|}{|X| + |Y|}$$

As is evident, it is not very different from the Jaccard index.

For dice coefficient, again w=0.2 produced the best results:

Score: 0.3333	rel=1	rank=1	qid=1	Classical music may never be the most popular music
Score: 0.2476	rel=1	rank=1	qid=2	Climate change and energy use are two sides of the same coin.
Score: 0.4615	rel=1	rank=1	qid=3	The best mirror is an old friend
Score: 0.1818	rel=1	rank=3	qid=4	If you see a friend without a smile, give him one of yours
Score: 0.1142	rel=1	rank=1	qid=5	Old friends are best

(MRR) Mean Reciprocal Rank ::0.8666666666666668

2 Error Analysis

Assuming best case analysis (a value of w which gives the best results) for each of the scoring metrics, it is evident from the results that the system is unable to rank the relevant document for query 4 at the top. All the other queries are able to retrieve the correct documents at rank 1. This trend is consistent among each of the three measures. A deeper look into the content of query 4 makes it clear that word level token matching or token matching in general is not enough to measure similarity. Feature vectors should be constructed using Machine Learning techniques and powerful semantic analysis. Thus, the problem is more of constructing appropriate feature vectors that represent the meaning of a document aptly, rather than choosing sophisticated similarity measures to compare those feature vectors. In other words, although a good similarity measure is important, how we construct and choose our features is much more important for improving system performance.

In case of queries 1, 2, 3 and 5, we just got lucky that simple token matching worked. In complicated information retrieval tasks, it would give a mediocre performance at best.