# 11-791 Design and Engineering of Intelligent Information System

Homework 4 Report

*Engineering and Error Analysis with UIMA*

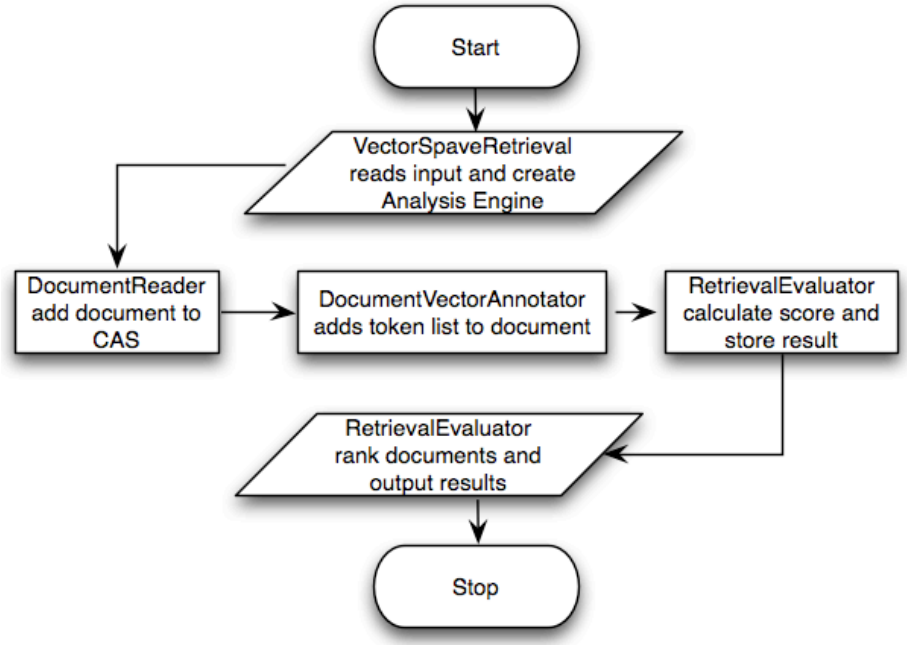
Name : Da Teng

AndrewID: dateng

# Task 1. Building Vector space Retrieval Model using UIMA

## 1.1 System architecture

This system works on the problem to match a question with many candidate answers. Candidate answer will be assigned a score and be ranked by that score as output. Following graph shows the system pipeline.




Since there is an existing archetype for this homework, what I mainly do is to modify and implement the methods in DocumentVecotorAnnotator and RetrievalEvaluator to make the system works like the graph above. In this homework, I didn't modify the typesystem or pipeline of the archetype, the major implementation is in class DocumentVecotorAnnotator and RetrievalEvaluator.

## 1.2 DocumentVectorAnnotator Design

DocumentVecotorAnnotator is responsible for annotating tokens in the documents.

- **process method**

This is the default method required by the analysis engine. It is a wrapper of all the mehods in DocumentVecotorAnnotator class, and will be called whenever a document is processed.

- **createTermFreqVector method**

For each document, we generate a token list containing all the tokens in it using the createTermFreqVector method in DocumentVecotorAnnotator class. And then we set this token list as the value of FSList attribute of its corresponding document, so that every document holds a token list of its own, which can be used in the scoring procedure.

One thing worth mentioning is that in order to increase the accuracy of the system, this method performs stemming, stopwords removal and converts all letters into lower case when it is annotating tokens. This is done by using an external class called EnglishAnalyzerConfigurable which is based on Lucene 4.3.0 toolkit.

- **stemToken method**

This method is used to perform stemming, stopword removal and convert letters into lower cases for a token. It is used by the createTermFreqVector method. It uses APIs from EnglishAnalyzerConfigurable, an external class based on Lucene 4.3.0 toolkit.

## 1.3 RetrievalEvaluator Design

As the CAS consumer of the analysis engine, this class is responsible for evaluating documents, recording the evaluated results and giving out the final aggregate output. RetrievalEvaluator assign a score to a document by comparing the similarity between that document and the query, and then score the qId, relevant value and evaluated socore of that document in local data structures. When system finish processing all the documents, RetrievalEvaluator will print out ranked information for all documents and compute the Mean Reciprocal Rank for all queries.

- **initialize method**

This method is part of the analysis engine that we can choose to implement. This method is called when a object of the class is created. In this method, local variables like qIdList and relList are initialized. These variables are used to record evaluated document results.

- **processCas method**

This is the default method required for a CAS consumer. It is called whenever a document is processed. In this method, the annotated result of a document is recorded. For example, the qId and token list will be added to local data structure of the RetrievalEvaluator class. But the score of a document has not been calculated in this class, in case that some queries could be read latter than its corresponding documents. The score of a document will be calculated when analysis engine finish reading in all the documents.

- **collectionProcessComplete method**

This method is only called once when the aggregate analysis engine comes to an end of the processing pipeline. When this method is called, the aggregate analysis engine has finished reading in all the documents and annotating tokens in these documents. In other words, the local data structures of the RetrievalEvaluator class holds information of all the input documents at this time.

So that in this method, I calculate scores for all the document and print the ranked documents by descending scores. At last calculate the Mean Reciprocal Rank for all queries and print it out. For this homework, I not only implement the cosine similarity method, but also implement the Dice coefficient method and Jaccard coefficient method. So there will be 3 sets of evaluation result. Each uses different method stated above.

- **computeCosineSimilarity method**

A method that computes the cosine similarity between the query and a document. The query and the document are represented as two Map that containing their tokens.

- **computeDiceCoefficient method**

A method that computes the Dice coefficient between the query and a document. The query and the document are represented as two Map that containing their tokens.

- **computeJaccardCoefficient method**

A method that computes the Jaccard coefficient between the query and a document. The query and the document are represented as two Map that containing their tokens.

- **sortAndPrintResult method**

Sort the evaluated document results in an ArrayList and print the sorted results.

- **evaluateByCosineSimilarity method**

A wrapper method that combines compute CosineSimilarity method and sortAndPrintResult method.

- **evaluateByDiceCoefficient method**

A wrapper method that combines compute DiceCoefficient method and sortAndPrintResult method.

- **evaluateByJaccardCoefficient method**

A wrapper method that combines compute JaccardCoefficient method and sortAndPrintResult method.

- **compute_mrr**

A method that compute the Mean Reciprocal Rank for all queries.

- **getQrySentIndex method**

A helper method that gets the query index in a local data structure (like ArrayList) for a corresponding document.

# Task 2. Error Analysis

## 2.1 Basic Evaluation Result Based on Cosine Similarity

If we use cosine similarity to evaluate the similarity between documents and query and don't do any processing on the tokens (no stemming, no stopwords removal, etc.), the evaluated results is as follows:

```
Ranked result evaluated by consine similarity:
Score: 0.4522670169  rank=1  rel=1 qid=1 Classical music may never be the most popular music
Score: 0.1020620726  rank=1  rel=1 qid=2 Climate change and energy use are two sides of the same coin.
Score: 0.5070925528  rank=1  rel=1 qid=3 The best mirror is an old friend
Score: 0.1721325932  rank=3  rel=1 qid=4 If you see a friend without a smile, give him one of yours
Score: 0.0000000000  rank=3  rel=1 qid=5 Old friends are best
 (MRR) Mean Reciprocal Rank ::0.7333333333333334
```

We can see that for query 1 2 and 3, the evaluated result is perfect, which has MRR=1.0. But for query 4 and query 5, the result is bad, which lower the MRR as a whole. System doesn't work well for the last two queries because there are not enough common tokens in the last two queries between their correct documents. If look into the last two quires, we can find that there are actually similar tokens that has not been recognized, for example, "friends" in query 4 and "friend" in its corresponding answer, "old" and "friend" in query 5 and "Old" "friends" in its corresponding answer.

## 2.2 Evaluation Result with Processed Token Based on Cosine Similarity

According to the analysis above, I process tokens in the sentence using stemming, stopword removal and changing all letters into lower cases to increase the common words between a query and its answer. And still evaluate the similarity between documents and query using Cosine Similarity. The result is shown below:

```
Ranked result evaluated by consine similarity:
Score: 0.5773502692  rank=1  rel=1 qid=1 Classical music may never be the most popular music
Score: 0.4330127019  rank=1  rel=1 qid=2 Climate change and energy use are two sides of the same coin.
Score: 0.5000000000  rank=2  rel=1 qid=3 The best mirror is an old friend
Score: 0.1360827635  rank=3  rel=1 qid=4 If you see a friend without a smile, give him one of yours
Score: 0.2357022604  rank=1  rel=1 qid=5 Old friends are best
 (MRR) Mean Reciprocal Rank ::0.7666666666666667
```

The overall MRR is a little bit higher this time because we rank the answer of query 5 correctly. However, the answer of query 3 is ranked a place lower this time. I think this is because cosine similarity doesn't penalize high frequency tokens so token "best" in the first-ranked answer carry a large weight in the numerator and causes the score of the sentence to over-weight the correct one. To solve this problem, we can try to use other methods to compute the similarity between two documents.

## 2.3 Evaluation Result with Processed Token Based on Dice Coefficient and Jaccard Coefficient

In this part, I try to use two other matrices, Dice coefficient and Jaccard coefficient, to evaluate the similarity between two documents. When evaluating the similarity between two documents, these two matrices counts the absolute value of common tokens between them, rather than using product of term frequency, so that high frequency tokens in documents are less likely to be overweighed. The result after applying these two matrices are shown below:

```
Ranked result evaluated by Dice coefficient:
Score: 0.4444444444  rank=1  rel=1 qid=1 Classical music may never be the most popular music
Score: 0.4285714286  rank=1  rel=1 qid=2 Climate change and energy use are two sides of the same coin.
Score: 0.5000000000  rank=1  rel=1 qid=3 The best mirror is an old friend
Score: 0.1333333333  rank=3  rel=1 qid=4 If you see a friend without a smile, give him one of yours
Score: 0.2222222222  rank=1  rel=1 qid=5 Old friends are best
 (MRR) Mean Reciprocal Rank ::0.8666666666666668


Ranked result evaluated by Jaccard coefficient:
Score: 0.2857142857  rank=1  rel=1 qid=1 Classical music may never be the most popular music
Score: 0.2727272727  rank=1  rel=1 qid=2 Climate change and energy use are two sides of the same coin.
Score: 0.3333333333  rank=1  rel=1 qid=3 The best mirror is an old friend
Score: 0.0714285714  rank=3  rel=1 qid=4 If you see a friend without a smile, give him one of yours
Score: 0.1250000000  rank=1  rel=1 qid=5 Old friends are best
 (MRR) Mean Reciprocal Rank ::0.8666666666666668
```

We can see that Dice coefficient and Jaccard coefficient has a better performance on evaluating the similarity between two documents. The system gets the best result for queries 1, 2, 3 and 5. Query 4 is a long a complex one, and it has not many common words with the correct answer, so it's hard to improve its performance.