FINAL REPORT

Name: Ningna Wang
AndrewID: ningnaw
Date: 10/21/2014

1. Design Aspect

Building Vector space Retrieval Model

In this task, I implemented a simple vector space retrieval system. I followed the pipeline offered by instruction document.

1.1 Collection Reader

The *DocumentReader* reads documents (small documents that each of which consists only one sentece) from CAS. Then, it recognizes query ID, relevance assessment and document text of each document and saves them into Document type.

1.2 Annotator

The DocumentVectorAnnotator uses a white-space tokenizer to split terms in each document. Then, I use hash map to calculate frequency of each term. And update the tokenList in CAS.

1.3 Cas Consumer

The RetrievalEvaluator is the most important part that I spent more time on implementing it. What we want to show in our output is the rank of cosine similarity of each relevant document in every query.

1. So I use hash map to store term-frequency vector. For all queries and relevant document, I use map to store their ID as well as term-frequency vector. For documents that are not relevant, I use arraylist to store all non-relevant documents of one query, then use hash map to store every query ID as well as its arraylist.
2. After calculating cosine similarity of relevant and non-relevant documents of each query, I use hash map to store query ID and arraylist that stores cosine similarities. Then sorting these values using Collections.sort() and save the rank of relevant documents into a new arraylist.
3. Last, I calculate MMR (Mean Reciprocal Rank) to evaluate its performance.

2. Error Analysis & System Improvement

In task1, I use a white-space tokenizer and cosine similarity to implement this code pipeline. Figure 1 shows the result of task 1. We can see that MMR is 0.4375, which is not that satisfying. So I consider many factors that may cause this low MMR.

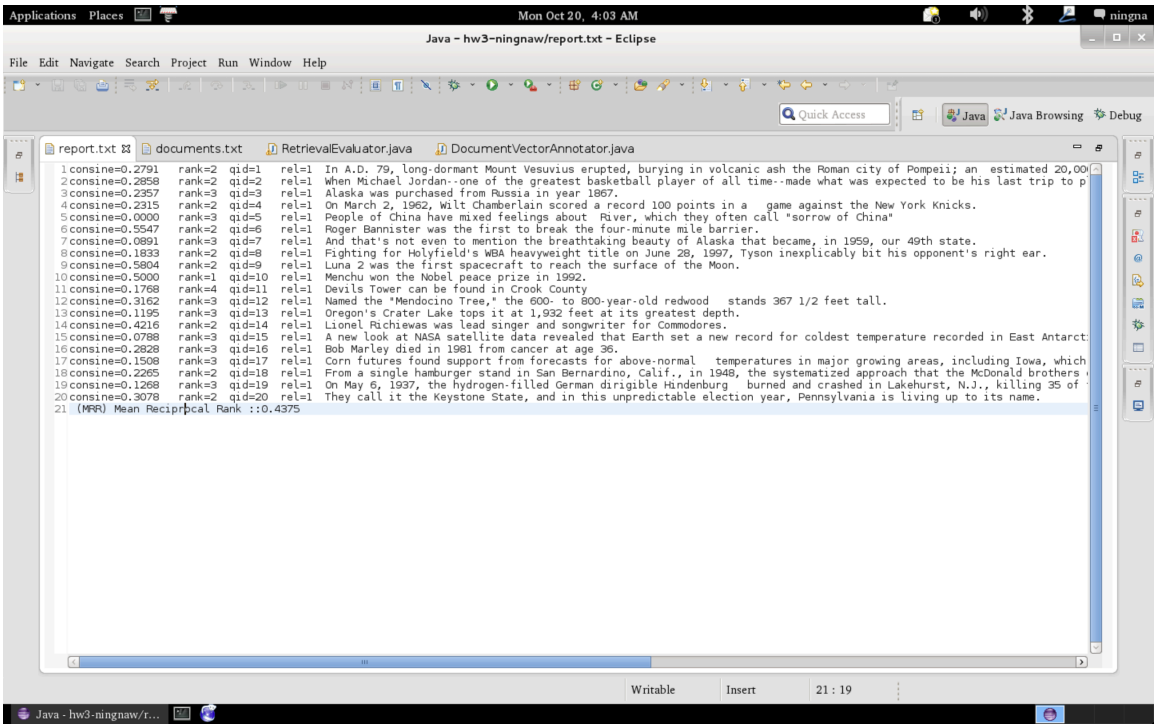

Figure 1. Original result

2.1 Bad tokenization algorithm

The tokenization algorithm I used in task 1 split sentence only by white space. This is not good since punctuations and symbols may not take into consideration. Here are examples that white-space tokenizer do not perform well.

qid	rel	error	correct
1	1	Pompeii;	Pompeii
2	1	time--made	time made
5	1	“sorrow	sorrow

Table 1.

In this case, tokening method should be improved. I use regular expression to improve tokenization algorithm, which take into account punctuations. Thus, sentences in document will be split on not only white space but also punctuations. Figure 2 shows the result of this improved algorithm. The MMR became 0.5375 from 0.4375(original result).

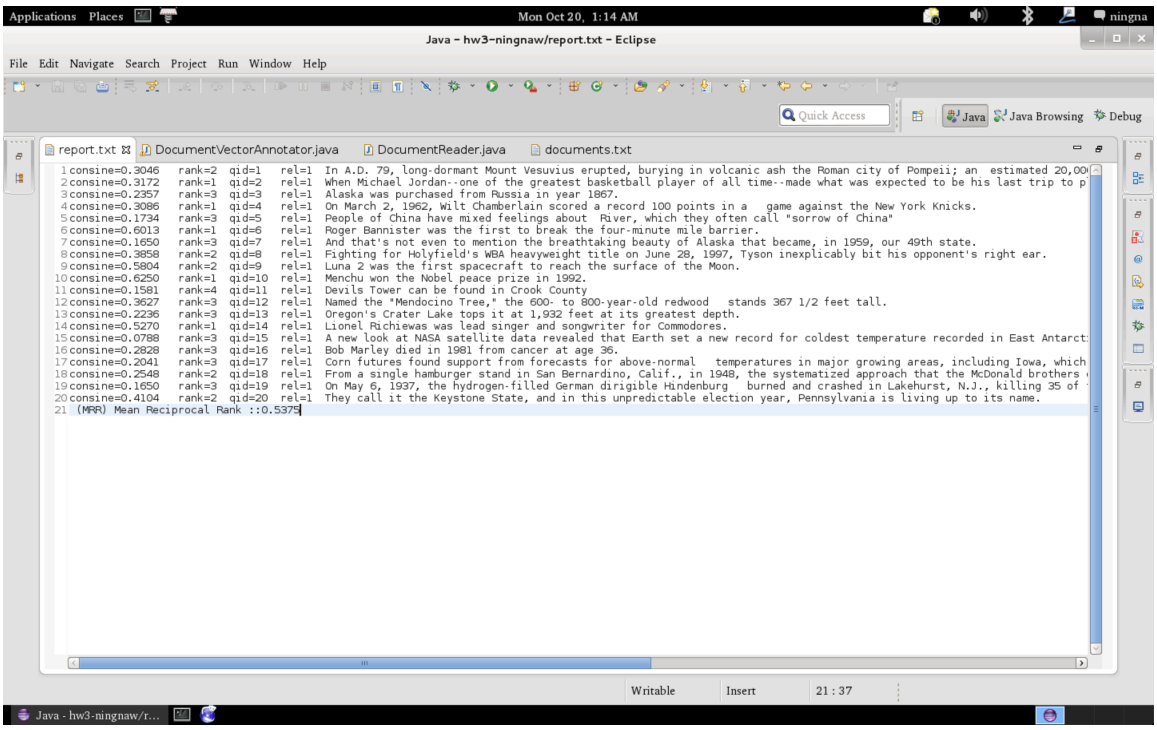

Figure 2. Result of new tokenization algorithm

2.2 Bad stemming algorithm

The simple vector model I used in task 1 cannot know that different tense of a same verb should mean the same thing. Here are examples in impute data that simple vector model do not perform well.

qid	rel	error	correct
8	99 and 1	'bite' not match 'bit'	'bite' and 'bit' have same meaning
16	99 and 1	'die' not match 'died'	'die' and 'died' have same meaning

Table 2.

In this case, simple vector model should be improved. I use Stanford lemmatizer to improve stemming algorithm. Figure 3 shows the result of this improved algorithm. After using this algorithm, the MMR became 0.55 from 0.4375(compared with original result, not after using new tokenization algorithm).

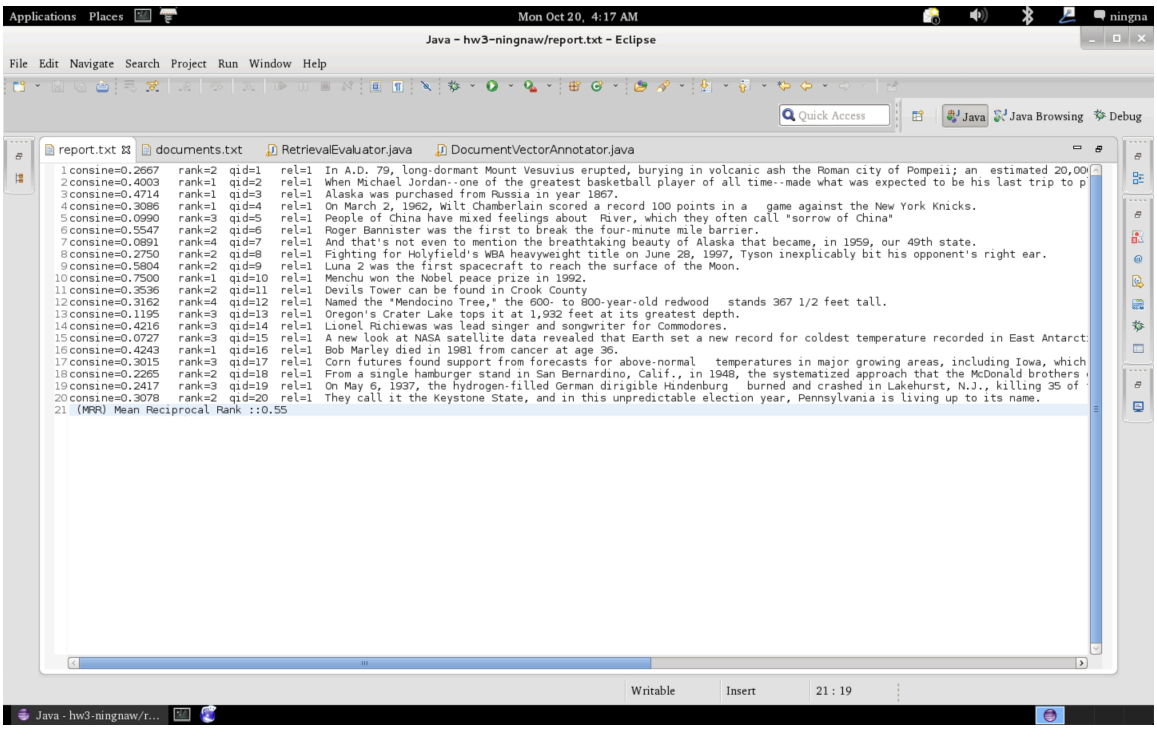

Figure 3. Result of new stemming algorithm

2.3 simple similarity measure

In task 1, the similarity measure I used is cosine similarity, which is a measure of similarity between two vectors of an inner product space that measures the cosine of the angle between them. If cosine similarity is expressed over bit vectors A(a_1, a_2, …, a_n) and B(b_1, b_2, …, b_n), then it can be written as:

$$F(A,B) = \frac{A \cdot B}{\sqrt{|A|^2} \times \sqrt{|B|^2}} = \frac{\sum_{i=1}^{n} a_i b_i}{\sqrt{\sum_{i=1}^{n} a_i^2 \times \sum_{i=1}^{n} b_i^2}}$$

However, I'm not sure that whether this method is good enough or not. So I implement other two similarity measure methods: Jaccard coefficient and Dice coefficient.

2.3.1 Jaccard coefficient

The Jaccard similarity coefficient is a statistic used for comparing the similarity and diversity of sample sets. If Jaccard similarity is expressed over bit vectors A(a_1, a_2, …, a_n) and B(b_1, b_2, …, b_n), then it can be written as:

$$F(A,B) = \frac{A \cdot B}{|A|^2 + |B|^2 - A \cdot B} = \frac{\sum_{i=1}^{n} a_i b_i}{\sum_{i=1}^{n} a_i^2 + \sum_{i=1}^{n} b_i^2 - \sum_{i=1}^{n} a_i b_i}$$

Figure 4 shows the result of using Jaccard instead of cosine similarity. After using it, the MMR became 0.4625 from 0.4375(implemented on task 1 and compared with original result). The MMR did not improve rapidly, but it still shows better result.

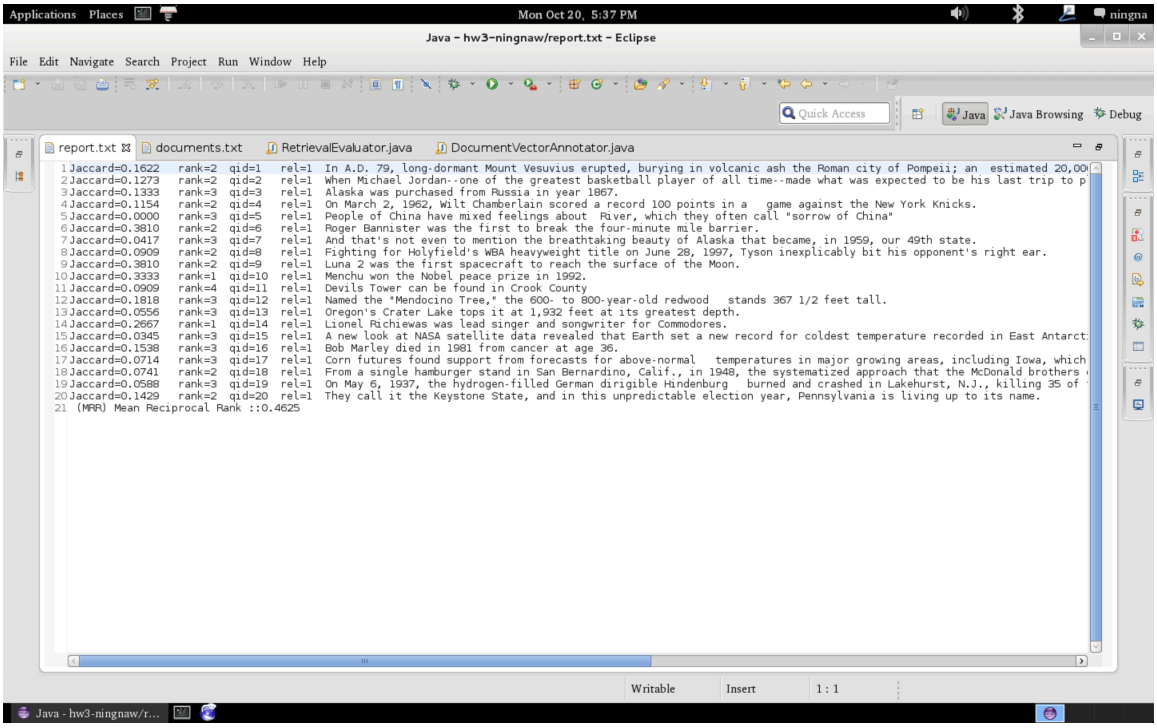

Figure 4. Result of Jaccard coefficient

2.3.2 Dice coefficient

Dice coefficient is also a statistic used for comparing the similarity of two samples. If Dice coefficient is expressed over bit vectors A(a_1, a_2, …, a_n) and B(b_1, b_2, …, b_n), then it can be written as:

$$F(A,B) = \frac{2(A \cdot B)}{|A|^2 + |B|^2} = \frac{2\sum_{i=1}^{n} a_i b_i}{\sum_{i=1}^{n} a_i^2 + \sum_{i=1}^{n} b_i^2}$$

Figure 5 shows the result of using Dice instead of cosine similarity. After using it, the MMR became 0.4625 from 0.4375(implemented on task 1 and compared with original result), which has same result with implementing Jaccard coefficient. Maybe it is because the sample input is typical in these two implementations.

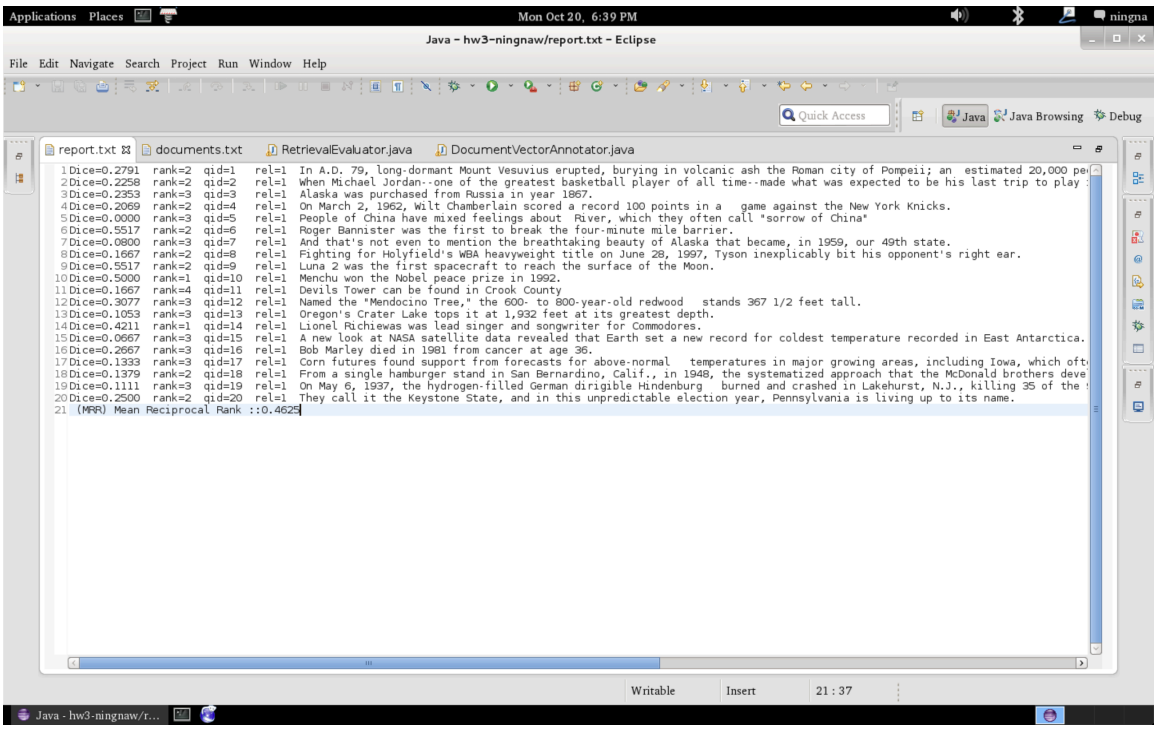

Figure 5. Result of Dice coefficient

2.4 Total System Improvement

Based on all individual improvement I did from 2.1 to 2.3, I came up with an idea that I can combine these improvements together. So I implement it. And result shows that this idea really performed well. It improved MMR from 0.4375 to 0.6833. And Figure 6 shows UML diagram of my implementation as well as Figure 7 shows result of this implementation.

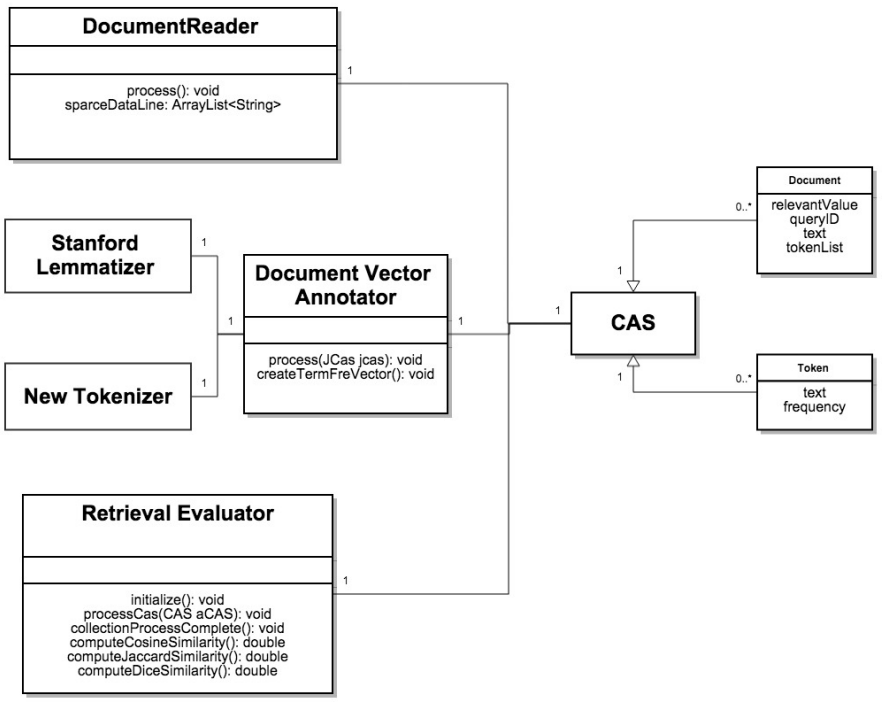

Figure 6. UML Diagram

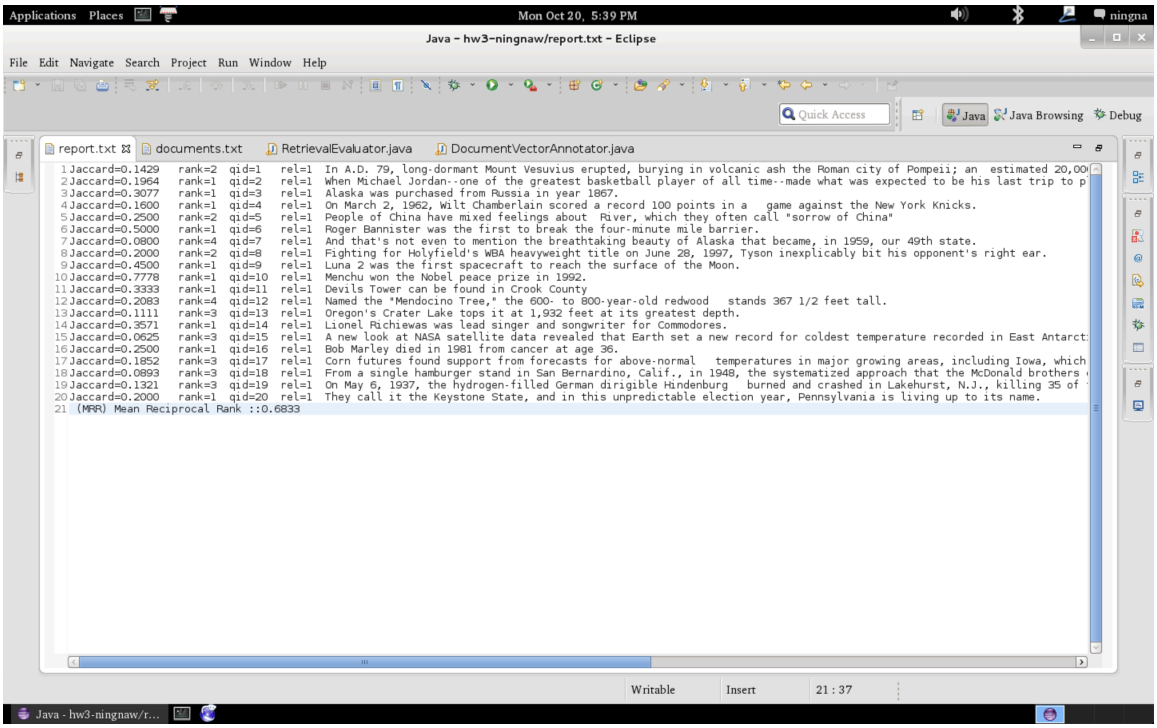

Figure 7. Result of Total Improvement