Report for Homework3: Xingyu Yan(xingyuy@andrew.cmu.edu)

Note: As task1 don't really have so much space you can design, this report will mainly talk about my design and implementation of task2.

1) Task1 Result Analysis:

QUERY: qid=1 rel=99 Give us the name of the volcano that destroyed the ancient city of Pompeii

FALSE: cosine=0.639602 rank=1 qid=1 rel=0 Vesuvius is located near the ruins of the destroyed city of Pompeii.

TRUE: cosine=0.279145 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.

This query mismatch is mainly caused by **vocabulary mismatch**: the program cannot map the word volcanic to word volcano. Using stemming algorithm will solve this problem. Also, another reason of this case in wrong order is that we **counted stop words as a match** such as "the" or "of", a stopword filter may help this.

QUERY: qid=2 rel=99 What has been the largest crowd to ever come see Michael Jordan

FALSE: cosine=0.300965 rank=1 qid=2 rel=0 A supposedly last play of Michael Jordan gathered some of the largest crowd in history of NBA.

TRUE: cosine=0.285774 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.

This query mismatch is mainly caused by the **vocabulary mismatch**, hopefully a new tokenize method will help.

QUERY: qid=3 rel=99 In which year did a purchase of Alaska happen?

FALSE: cosine=0.421637 rank=1 qid=3 rel=0 William Seward negotiated a purchase of Alaska for $7.2 million.

FALSE: cosine=0.267261 rank=2 qid=3 rel=0 1867 - U.S. President Andrew Jackson proclaims treaty for purchase of Alaska from Russia.

TRUE: cosine=0.235702 rank=3 qid=3 rel=1 Alaska was purchased from Russia in year 1867.

In this case, **term weight** can be the problem. If we want the relevant document to go first, we should put in an NLP tool so that we can understand that the "year" is the main target we are looking for and rank the particular term a higher weight. Also the **term understanding** is a issue here as well: for the keyword "year", we may count the year-like string as a match. *(In fact I think in this case the 2nd mismatch can be accepted as a true answer.)*

QUERY: qid=4 rel=99 What year did Wilt Chamberlain score 100 points?

FALSE: cosine=0.273861 rank=1 qid=4 rel=0 A 100 point game was a highlight in a career of Wilt Chamberlain

TRUE: cosine=0.231455 rank=2 qid=4 rel=1 On March 2, 1962, Wilt Chamberlain scored a record 100 points in a game against the New York Knicks.

Issue: 1) **vocabulary mismatch** on scored to score, 2) **term understanding** on the year-like string to year.

QUERY: qid=5 rel=99 What river is called China's Sorrow?
FALSE: cosine=0.408248 rank=1 qid=5 rel=0 Yellow river is often called the mother of China
FALSE: cosine=0.163299 rank=2 qid=5 rel=0 Yangtze is longest river in Asia in general and in China, in particular.
TRUE: cosine=0.000000 rank=3 qid=5 rel=1 People of China have mixed feelings about River, which they often call "sorrow of China"

Issue: **term understanding** on different expression.

QUERY: qid=9 rel=99 What was the first spaceship on the moon
FALSE: cosine=0.652929 rank=1 qid=9 rel=0 Eagle was the first manned spacecraft that reached the surface of the moon
TRUE: cosine=0.580381 rank=2 qid=9 rel=1 Luna 2 was the first spacecraft to reach the surface of the Moon.

Issue: **term understanding**: an additional restriction made it not the right answer

The basic types of error that affects the result in 10 failed queries are:

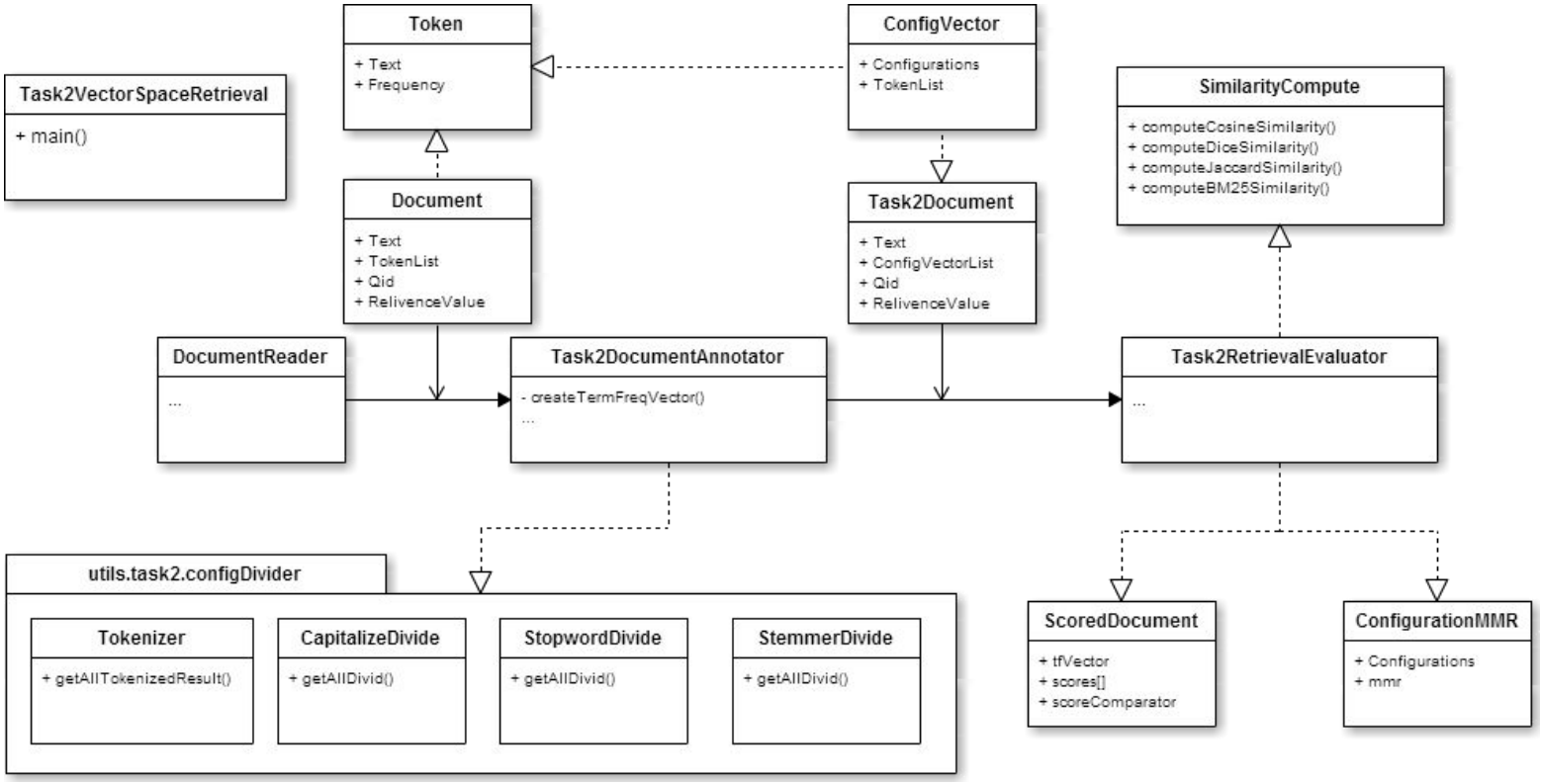
Vocabulary mismatch	Symbol contained in word	3
	Different word form	2
Term understanding	Term expansion (year -> 1962)	4
	Term expressing	2
	Restriction	1
Term weighting	Stop words overweight	1
	Key words underweight	3

2) Task2 System design:

Objective: My system design is targeting to **automatic generate report** for every method that mentioned in the homework design, and find the best combination targeting this particular data-set. Although I know this design are still miles far from an elegant design, it now can provide enough help on debug and error analysis, and giving I have spent over 20 hours on this, I think I have to stop here.

Usage: Same way of triggering task1 (same path too) except set the mainClass = Task2VectorSpaceRetrieval.java. It's report is **task2_report.txt**, and I appended an excel file in the doc directory which is the mmr matrix for every method combination that extracted from report.

Current UML Diagram:



Brief Work Flow:

Using Task2VectorSpaceRetrieval as main class, the only different form the one provided in task 1 is different descriptor path.

I kept **DocumentReader**, and so as the original TypeSystem its using, but in **Task2DoocumentAnnotator**, instead of package the text in to one single tokenList, applying different tokenization setting, I generated a list of token list (which is essentially **Task2Document**). And in order to keep track of all their performance, I packed the configuration of how they are generated in to the type system, which is **ConfigVector**.

Those different configurations included:

a. Using naive tokenizer (white space), replacing some given symbols to space or even eliminate symbol and single characters(like change China's to China);
b. Do nothing or converting all characters to lowercase;
c. Do nothing or using stopword list to filter stopwords.
d. Do not stem, using given Stanford stemmer or using apache snowball stemmer.

All of these configurations are implemented in **utils.task2.configDivider**, which contained a series of tool class and provided 36 different output tokenList from one single document.

In the **Task2VectorSpaceRetrieval**, I reconstructed the data structure from the original grouped by document to a structure grouped by configuration so that it can calculate the MMR of each algorithm of each configuration. TF-IDF is the alternative term frequency so I implemented inside **Task2VectorSpaceRetrieval**, and packaged all the similarity algorithm to a util class named **SimilarityComput**.

In **SimilarityComput,** I implemented cosine similarity, Dice similariy, Jaccard similarity and BM25. And for the BM25 algorithm, I tried several times about changing both k1 and b in it but it doesn't make so much difference, and eventually I choose **k1=1.2** and **b = 0.75** as the final

algorithm configuration for BM25.

And because all other algorithms required original term frequency, I only used calculated TF-IDF to calculate the naive cosine similarity, so that for each configured tokenized document, I'll generate 5 MMR so that its totally 180 MMR result. I stored them in **ConfigurationMMR** along with all their configurations including tokenization and similarity calculation.

At last, I sorted the **ConfigurationMMR** list and print out the top 10 configurations of this data-set on the console, a

nd kept all the processing data in the report file.

3) Task2 Result Analysis:

In the result matrix, we can see that the best results are almost all coming from the naive cosine similarity, but in the overall tabs cosine similarity didn't have an substantial advance over other algorithms (only 0.5% over TFIDF, 1.5% over Dice), which in my interpretation we can not be sure that's the better way to do it due to the limited test sample size.

And as for tokenization methods, we can see that **filter symbols** got an noticeable improvement over the white space tokenizer (7%), and so is **using stemmer** over not using any stemmer(8%). **Filter stopwords** provided only 0.8% improvement on average, which is not quite satisfying, I guess maybe a better stopword list may help. **Unified lower case** contributed 3% on average.

The most interesting part is, the top 5 results are presenting a very promising pattern: they didn't care about case unify(false:true = 2:3), they are more likely to **avoid** using **stoplists** which advanced on average(false:true = 4:1), most of them are coming from **naive cosine similarity** (naive cos : tf-idf cos = 4:1), and most importantly, all of them are stemmed with **Stanford stemmer**. Looking like those Stanford libraries are really handy....

Again, the detail of the result are provided in an excel table in the doc directory. As for the results are not entirely what I expected, just hope that it didn't went wrong.