11-791 Design and Engineering of Intelligent Information System
Assignment 4

Laleh Roosta Pour, AndrewID: lroostap

1. Task 1

First, I imported the project and created the proper maven project. I didn't make any change in the collection reader and typesystem.The two main files I was working on in this assignment are "DocumentVectorAnnotator.java" and "RetrievalEvaluator.java".

First, I implemented the required code in "DocumentVectorAnnotator.java" to update the tokenList of Document and the CAS. The in this class extracts the bag of word feature vectors from the text sentences. It means the term and term frequency are filled with the word and word occurrence for that specific sentence. In the process of improving the system I modified this class and added new method for generating the features. There are 3 ways in this class, the corresponding part of the code for each strategy is marked with the proper comments. In the last submitted version of the assignment the two first ways are commented out and the third one is applied. I will explain the methods later. I used a part of my code from previous home work for using the stanford tokenzier to generate tokens. I added the stanfordCoreNLP dependency to the "pom.xml".

Second, the necessary code was implemented in "RetrievalEvaluator.java" which is a CAS consumer class in order to calculate the cosine similarity between query sentence and each of the subsequent document sentence and the (Mean Reciprocal Rank) MRR metric for the retrieval system. I created a class DocInf in the similar package with "RetrievalEvaluator.java" that which records queryID, cosineSimilarity, relValue, rank, etc. I implemented a comparator method for this class to be able to sort DocInf objects based on their cosineSimilarity values.

2. Task 2

The first method (strategy-1) that I used to create the terms was splitting the text by spaces and then counting the frequency of each term. Considering this as a baseline, the performance of the system is as follow:

```
Score: 0.45227  rank: 1   rel: 1   qid: 1

Score: 0.30619  rank: 1   rel: 1   qid: 2

Score: 0.50709  rank: 1   rel: 1   qid: 3

Score: 0.17213  rank: 3   rel: 1   qid: 4
```

```
Score: 0.15811  rank: 1   rel: 1   qid: 5

Mean Reciprocal Rank (MRR)::0.87
Total time taken: 0.744
```

Since I was using the Stanford tokenizer in previous homeworks, I also tested the system by generating tokens by that tokenizer (strategy-2) and result is as follows:

```
Score: 0.45227  rank: 1   rel: 1   qid: 1

Score: 0.29417  rank: 1   rel: 1   qid: 2

Score: 0.46291  rank: 2   rel: 1   qid: 3

Score: 0.25000  rank: 3   rel: 1   qid: 4

Score: 0.15811  rank: 1   rel: 1   qid: 5

Mean Reciprocal Rank (MRR)::0.77
Total time taken: 0.805
```

The third strategy (strategy-3) is combination of lemmatization and application of stopwords which has the following result:

```
Score: 0.61237  rank: 1   rel: 1   qid: 1

Score: 0.43301  rank: 1   rel: 1   qid: 2

Score: 0.44721  rank: 2   rel: 1   qid: 3

Score: 0.33806  rank: 1   rel: 1   qid: 4

Score: 0.47140  rank: 1   rel: 1   qid: 5

Mean Reciprocal Rank (MRR)::0.9
Total time taken: 2.367
```

In order to make the base line I used `strategy-1` and `strategy-2` to see which one works better. The result showed that generating the tokens using the Stanford tokenizer, decrease the performance compare to simple split by space. The reason is that the characters like "'" or "," will be considers as a token in the tokenizer, while splitting the text by space leave them next to

the adjacent (I meant the word beside them without the space). Therefore, the number of tokens increases, but while they are not similar to the query which increase the denominator of cosine similarity formula which results in the lower score. The score changed the ranking and as a result the MRR decreases. For example, in query number three, we have "One's" by using the toknezire we have three tokens "One" and "'" and "s", however with space splitting we will have only one token "One's".

Considering the query, other sentences and the result, we can see some general issues. First is the presence of stopwords which can confuses and leads the result in the wrong way. These words might increases the cosine similarity for the wrong reason when two vectors has many similar stopwords, when doesn't really mean they are similar. On the other hand, they can cause the low cosine similarity when one vector has many of them while the other one doesn't have and they don't really represent the similarity between those two vectors. Therefore I put a string including the stopwords in the stopword file splitting by the space. Then, when I want to fill the tokens and frequencies, I ignore the ones in the stopwords.

The other problem are the words in various forms, which are not exactly the same, but their stem/lemma are the same. For example the plural form of a noun, the different conjugation of a verb, etc. If we use the lemma form of these words, we can probably get a better (precise) cosine similarity. For sure it depends on the data set and this also could cause some problem, but in general, stems and lemmas are very helpful in nlp projects. For example in this document we see that the word "friend" and "friends" were repeated many times, but without lematization, we won't be able to capture the similarity based of them. Therefore I added the StanfordCorenlp model and use it to generate lemmas. I added the related code for using the tool in their website:

`http://nlp.stanford.edu/software/corenlp.shtml`

The result shows applying stopwords strategy and representing the terms as lemmas improved the ranking for query number 4 and increases the MRR to 0.9.

There are many other options that I think can be very helpful like creating a dictionary which has the synonym and antonym of the word, abbreviates, etc. Similar to lemma there is also a method to consider the words only with certain number of letters which might also improve the result. I didn't have enough time to test all these ideas.