Design and Engineering of Intelligent Information Systems (DEIIS)

Homework #4

Weston Feely
andrew ID: wfeely

October 28, 2013

System Design

For this assignment, I filled out the missing code in the DocumentVectorAnnotator annotator Java class and RetrievalEvaluator cas consumer Java class. This included re-purposing some code from my TokenAnnotator from previous homeworks to extract tokens and fill in the tokenList for each document, within the DocumentVectorAnnotator class. I did this using an arrayList and the provided methods from Utils.java to convert back and forth from arrayList to FSList.
Then, I edited RetrievalEvaluator by creating a new global variable, which was the vocabulary for all documents in the document set. I added up the frequencies of each word for the vocabulary vector and simultaneously created a term frequency (tf) vector for each document, by adding up the frequencies for each document and placing them into a separate HashMap.
Then, I needed more structure to represent the data pertaining to each document, and because I didn't know how to include Java objects such as HashMaps in the UIMA type system provided, I made my own "Doc" Java class, which houses the information for each document. This includes the text, queryID, relevance value, the frequency vector, and the cosine similarity to its query. I also created a "querySet" Java class to represent the sets of documents for each query. Each querySet object contains an arrayList of Doc objects for the documents, a single Doc object for the query, another arrayList of Doc objects for the ranked documents, and the queryID for the query. I then made a global arrayList of querySet objects, so I could loop through each querySet. While looping through the documents, I populated

each Doc object with its information and then put each document into its querySet, along with its corresponding query.
At this point, I also started calculating the idf scores for each document and word in the vocabulary. I made a global variable to contain the idf scores. Then, I calculated the tf and tf-idf scores for each document, using the tf vectors I had already put together and by multiplying together my tf-idf scores. I then calculated cosine similarity between each query and its corresponding documents, ranked the documents in the querySet by filling in the ranking Doc object arrayList that I had created for the querySet, using a selection sort on the cosineSimilarity of each document to its query.
At this point, I had a functioning system with a very low MRR. I realized I had made several mistakes in my system in my tf and idf calculations. I corrected these by referring to the forumlae on the wikipedia articles on tf-idf [2] and vector space model for cosine similarity [2]. At this point, I reached what I consider to be my baseline score of 0.6 MRR, using the "augmented frequency" calculations for tf from wikipedia's tf-idf page.

I improved upon this baseline by first making some small adjustments to my DocumentVectorAnnotator. I lowercased each token as it was read in and used the provided stoplist to ignore stopwords when filling my document frequency vectors. This brought my MRR up to 0.7. I then tried out the other tf calculation suggestions on the tf-idf wikipedia page. The raw frequency (raw counts for tf weights) version actually performed better than the more complex "augmented frequency" weights that I had initially tried out, with a MRR of 0.8. The log-scaled frequency weights ($log(f+1)$) provided the same MRR of 0.8. Finally, I tried the boolean "frequency" suggested by the wikipedia page, which gives a frequency of 1 for each term that exists in the document and 0 for each term that does not exist in the document. This boolean weight brought my MRR up to 0.9, still using the cosine similarity metric from the baseline system.
I had one further idea to improve my system, which was a little more creative. After lowercasing the tokens and applying the stoplist to remove stopwords from my document tf vectors, I further modified my tokens by making the token text "SHORT" for tokens with a length of 3 characters or less. This improvement brought my MRR up to 0.9, no matter which tf calculation strategy I used. I considered this to be an improvement, because I could use raw frequency counts and still have a 0.9 MRR. My idea behind this improvemt was that I could create "classes" for tokens based on their length and after experimenting a little I found that keeping the short tokens in the same "class" could improve the system, but keeping longer tokens

together would not.

Below is the final output of my system :

Output:

```
Score: 0.3925430411579019 rank=1 rel=1 qid=1 Classical music may never be the most
Score: 0.45113879909288235 rank=1 rel=1 qid=2 Climate change and energy use are tw
Score: 0.18595069834930666 rank=1 rel=1 qid=3 The best mirror is an old friend
Score: 0.1466611720754116 rank=2 rel=1 qid=4 If you see a friend without a smile,
Score: 0.009147995781601418 rank=1 rel=1 qid=5 Old friends are best
 (MRR) Mean Reciprocal Rank ::0.9
Total time taken: 0.65
```

References:


```
[1] http://en.wikipedia.org/wiki/Tf-idf
[2] http://en.wikipedia.org/wiki/Vector_space_model
```