11-791: Design and Engineering of Intelligent Information Systems

Homework #4 – Lars Mahler – lmahler@andrew.cmu.edu

Engineering and Error Analysis with UIMA

Task 1.1 – 1.3

In implementing this assignment, I used the archetype files as a starting point. I left the high-level structure (components, type system) unchanged: they seemed well-structured, and the results of the error analysis (see below) did not require changes at an architectural level. I did, however, make the following modifications:

- **DocumentVectorAnnotator**
  - **createTermFreqVector**: Implemented code to create the term frequency vector for each document. This code counts each token (and its frequency), creates token annotations, and then updates the source document with a list of tokens encountered in that document.
- **RetrievalEvaluator**
  - **processCas**: Implemented code to store the results of each document into a global list (as the document is processed during the processCas call). This list is then used later on during the **collectionProcessComplete** call, after all documents have been processed through the pipeline.
  - **collectionProcessComplete:** implemented code to calculate cosine similarity (query sentence vs. candidate answer sentence), rank candidate answer sentences, calculate the MRR, and then output the results to screen. Calculations of cosine similarity and MRR were encapsulated in the helper functions **computeCosineSimilarity** and **compute_mrr**.

Task 2: Error Analysis

The first round of implementation, using the test sentences provided, I obtained the following results:

- Question: 1
  - Score: 0.45226701686664544	Rank: 1 Rel: 1
  - Score: 0.35355339059327373	Rank: 2 Rel: 0
  - Score: 0.2672612419124244	Rank: 3 Rel: 0
- Question: 2
  - Score: 0.30618621784789724	Rank: 1 Rel: 1
  - Score: 0.0	Rank: 2 Rel: 0
  - Score: 0.0	Rank: 3 Rel: 0
- Question: 3
  - Score: 0.5070925528371099	Rank: 1 Rel: 1
  - Score: 0.4338609156373123	Rank: 2 Rel: 0

    - Score: 0.18257418583505536 Rank: 3 Rel: 0
- Question: 4
    - Score: 0.28867513459481287 Rank: 1 Rel: 0
    - Score: 0.22360679774997896 Rank: 2 Rel: 0
    - Score: 0.17213259316477406 Rank: 3 Rel: 1
- Question: 5
    - Score: 0.15811388300841897 Rank: 1 Rel: 1
    - Score: 0.11952286093343936 Rank: 2 Rel: 0
    - Score: 0.05773502691896257 Rank: 3 Rel: 0
- Mean Reciprocal Rank (MRR)::**0.8666666666666668**
- Total time taken: **1.039 seconds**

As you can see, on this limited test set, the pipeline worked pretty well, achieving .86 MRR. The only case that failed was question 4, which had the following question and answers (ranked in order of their original ranking):

- qid=4 rel=99 The shortest distance between new **friends** is **a smile**
- qid=4 rel=0 Wear **a smile** and have **friends**; wear **a** scowl and have wrinkles
- qid=4 rel=0 Behind every girls **smile** is **a** best friend who put it there
- qid=4 rel=1 If you see **a** friend without **a smile**, give him one of yours

The first thing I noticed was that certain words were not matching because the word forms in the query ("friends") were different from the correct answer ("friend"). As a result, I incorporated the Stanford Corenlp tokenizer into the **DocumentVectorAnnotator** stage of the pipeline, and used its lemmatizer feature to transform tokens to their lemmatized forms.

The second thing I noticed was that some of the words that caused incorrect sentences to move to the top of the rankings were words such as "a" – i.e. unimportant words that did not add semantic value. As a result, I incorporated a stoplist into the **DocumentVectorAnnotator** stage of the pipeline. Although this stoplist is defined within DocumentVectorAnnotator, it could be parameterized if the whole VectorSpaceRetrieval pipeline were encapsulated in an AE, AAE, or CPE.

When I reran the new, updated pipeline (using a stoplist of: {"be", "have", "to", "from", "with", "of", "the", "a", "an", "it", ",", ";"}), I obtained the following results:

- Question: 1
    - Score: 0.5773502691896257 Rank: 1 Rel: 1
    - Score: 0.4364357804719848 Rank: 2 Rel: 0
    - Score: 0.3481553119113957 Rank: 3 Rel: 0
- Question: 2
    - Score: 0.35856858280031806 Rank: 1 Rel: 1
    - Score: 0.0 Rank: 2 Rel: 0
    - Score: 0.0 Rank: 3 Rel: 0

- Question: 3
    - Score: 0.5163977794943222 Rank: 1 Rel: 0
    - Score: 0.4472135954999579 Rank: 2 Rel: 1
    - Score: 0.4472135954999579 Rank: 3 Rel: 0
- Question: 4
    - Score: 0.2721655269759087 Rank: 1 Rel: 0
    - Score: 0.25819888974716115 Rank: 2 Rel: 1
    - Score: 0.23570226039551587 Rank: 3 Rel: 0
- Question: 5
    - Score: 0.47140452079103173 Rank: 1 Rel: 1
    - Score: 0.33333333333333337 Rank: 2 Rel: 0
    - Score: 0.2886751345948129 Rank: 3 Rel: 0
- Mean Reciprocal Rank (MRR)::**0.8**
- Total time taken: **2.954 seconds**

As you can, this boosted the performance of question 4, however it lowered the performance of question 3. The overall effect was to lower the MRR, from .86 to .8. The processing time increased from ~1 second to almost 3 seconds. Despite the apparent decrease in performance, I actually think that this pipeline is an improvement, for the following reasons:

- This is a very small test set, with some very strange queries and answers – on a larger test set, with more representative queries and answers, lemmatizing and adding stopwords should (on average) lead to better MRR performance.
- Questions 3 and 4 are somewhat nonsensical (i.e. their rel=1 answers don't really make sense as answers to the query), and not representative of normal queries and answers – we probably shouldn't put too much weight on the importance of these sentences when evaluating performance.
- The fact that the processing time increased is not a problem: this reflects time needed to load Stanford Corenlp, and that cost would get amortized (to a very small amount) over a larger volume of queries and answers.