Text Technologies for Data Science: Assessment 2

s1107496

October 20, 2014

1 Introduction

This report describes the implementation of two information retrieval algorithms, brute.py and index.py, as well as refinements to the latter as implemented in best.py.

2 brute.py and index.py

2.1 brute.py

In brute.py, each news story was first transformed to lowercase and tokenized on whitespace, following which the tf-idf weighted cosine denominator component for the story was computed. Then, for every prior story, the tf-idf weighted cosine was calculated, drawing from both a pre-computed dictionary of token idf scores and a list-based cache of tokenized story/tf-idf weighted cosine denominator component tuples. In this way, only the query-document-unique numerator of the tf-idf weighted cosine formula needed to be calculated (for all tokens in both the query and document). If the score of the best matching document exceeded the given threshold, the result was written to a file and immediately flushed. Finally, the token/denominator tuple for the latest news story was added to the aforementioned cache.

2.2 index.py

index.py improved upon brute.py by implementing a term-at-a-time indexing. The index was represented as a dictionary with tokens as keys and lists of story number/token count tuples as values. As in brute.py, tf-idf weighted cosine denominators were also cached for each story. When calculating tf-idf weighted cosine, partial story scores were stored in a dictionary. For 10,000 stories, index.py ran roughly 3.5 times faster than brute.py.

3 best.py

best.py included a number of refinements over index.py, as listed below. For 10,000 stories, best.py ran 17.7 times faster than index.py (62.8 times faster than brute.py). As measured against the provided pairs.ref, best.py has an accuracy of 91.13%.

3.1 Stop word removal

After initial tokenization using the same regular expression as overlap.py, all tokens from the English-language stop word list provided by `nltk` were removed. This was performed for both queries and documents, and increased average precision by 0.0088.

3.2 Identical story checking

To account for morphological variation, each token was broken into constituent n-grams, which were then themselves added to the token list. This procedure was performed for both queries and documents. As

stated in the 29th September, 2014 Text Technologies lecture, character n-grams of length 4 or 5 work best for European languages - in this case, 4-grams yielded an average precision advantage of 0.003 over 5-grams. Adding both 4- and 5-grams decreased average precision. Overall, this method increased average precision by 0.0066.

3.3 Python-specific improvements

In both documents and queries, to capture meaningful pairings of words, bigrams from the stop-word-sanitized token list were themselves added to the token list. This improved average precision by 0.0033, and was found to be more effective than trigrams or both bigrams and trigrams.

4 Comparison of algorithm running times