11-791: Homework 3

Maya Tydykov

Tuesday, October 14, 2014

1 Error Analysis of Baseline System

The baseline system achieves an MRR of 0.4375. Only one query gets a relevant document ranked in first place; all other relevant documents lose out to irrelevant ones.

There are several pitfalls with the baseline version of the system. Error analysis shows that these pitfalls can be split into several error categories, including vocabulary mismatch (which can further include the category of lack of stemming), problems due to case sensitivity, and tokenization. An analysis of the output of the system based on these error classes reveals the following:

	Vocabularly Mismatch	Case sensitivity	Tokenization
# sentences	12/19	2/19	7/19

Furthermore, 5 of the 12 sentences that had a vocabulary mismatch had mismatches that could be solved by stemming.

Vocabulary mismatch is a class for cases where relevant sentences were missed because an important word in the query was either written in a different form or a different word was used entirely in the document sentences. Case sensitivity concerns cases where the query contains important words that are identical to words in documents except for a mismatch in case. Tokenization concerns cases where the the query contains an important word that would not have been matched against the same word in the document only because of a tokenization problem, such as not separating a word ending in a period, comma, or question mark, or a word that ends in an apostrophe and an 's'.

These problems can be fixed relatively simply. While it may not be easy to fix vocabulary mismatch as an overall category (unless one were to include synonmys, etc, which brings into question problems such as word

sense disambiguation), it is straightforward to fix this by using a stemmer and rather than storing and counting the full form of a word, only to store and count its stemmed form in the term vector. Case sensitivity can be fixed easily be converting everything to uppercase or lowercase, but this may result in false positives sometimes (such as when a proper name, lower-cased, matches a different, common word). A simple way to fix the tokenization problem is to separate punctuation from words.

Besides the problems described in the above error categories, it is generally not ideal to use cosine similarity with just term counts to rank documents the way it is implemented in the baseline. One pitfall is that this measure does not take into account non-topic, or common, words, particularly stop words. This includes words such as variations on the verb "to be", pronouns, determiners, etc. One way to fix this problem would be to use tf-idf instead of just raw term counts when calculating cosine similarity.

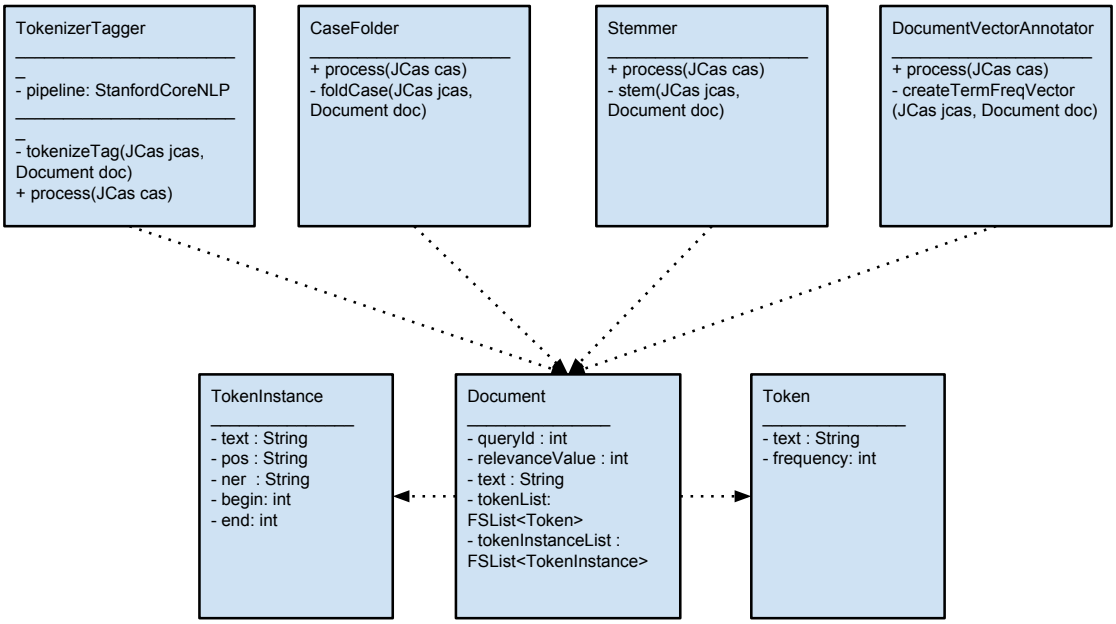
2 Improved Pipeline

Based on the above error analysis, the new system pipeline incorporates the following imporvements:

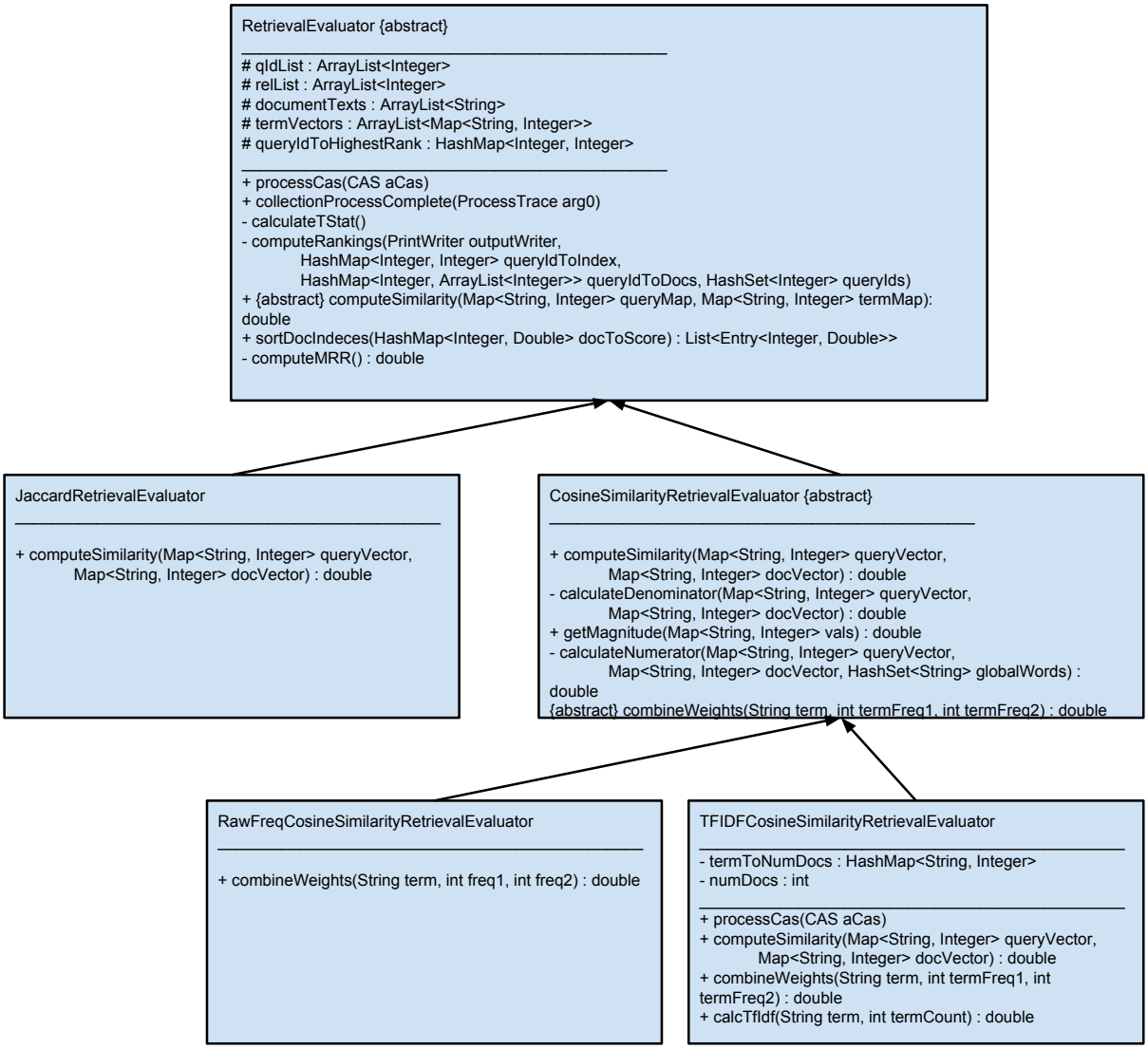
1) Using the StanfordCoreNLP annotator to tokenize based on punctuation as well as whitespace.
2) Using the StanfordCoreNLP annotator to get part of speech tags.
3) Lowercasing words that are not marked as proper nouns.
4) Using Stanford's Stemmer module to break non-proper nouns down to their stems for constructing term vectors.
5) Trying a TF-IDF weighting scheme for cosine similarity. TF-IDF is defined as: $\frac{tf(t,d)}{idf(t,D)}$, where $tf(t,d)$ can be most simply the raw frequency of term t in document d, and $idf(t,D) = log\frac{N}{|d \in D : t \in d|}$. (see Wikipedia entry).
6) Trying the Jaccard similarity coefficient as an alternative similarity metric. This is defined as: $J(A,B) = \frac{|A \cap B|}{|A \cap B|}$. (see Wikipedia entry).

The new system design can be seen in the following UML diagram, where only the most informative and important fields and methods are shown.

Preprocessing



Ranking



As illustrated in the above UML diagram, there are four annotator components in the preprocessing stage. TokenizerTagger uses StanfordCoreNLP to tokenize the input text and gather information about it such as each token's part of speech and named entity annotation. This information is then used to populate the document's TokenInstance list, which is a list of information about every individual token in the text, where tokens are made distinct by their position in the text.

Once a list of TokenInstance objects has been acquired, various annotators can be used to add information to the tokens or change their form. The two included in the pipeline are CaseFolder, which changes all token instances to lowercase as long as they are not marked as proper nouns, and Stemmer, which uses the Stanford Morphology class to reduce token instances to their stems.

After all processing on TokenInstances is complete, DocumentVectorAnnotator, populates the Document object's Token list, which collapses token instances with identical texts and keeps their frequencies.

After the term vectors have been constructed, the system moves into the document ranking stage. This part of the processing uses inheritance to unite common functionality while allowing experimentation with variations on ranking methods. Specifically, the common functionality for all ranking methods is defined in the RetrievalEvaluator abstract class. This class aggregates information commonly needed by all subclasses into the five fields, as shown in the UML diagram. In a larger, more extensive system, the information stored in these fields would be written out to files and read back in during the final processing step rather than being stored in memory.

Common functions for all subclasses of RetrievalEvaluator are computing the rankings, computing the similarity between a query and a document, sorting the document indeces by score, and computing the MRR.

There are two current overall variations on ranking schemes. One is the JaccardRetrievalEvaluator, which computes the similarity between a query and a document based on the above definition. The other is CosineSimilarityRetrievalEvaluator, which is another abstract interface that defines the general structure for computing the cosine similarity but leaves the weighting of the terms of the document and query in the numerator to the classes that extend it. There are two such classes currently defined. One uses the same method as the baseline (the raw frequency of terms). The other uses TF-IDF (defined above).

3 Experimental Results

I used Apache Commons' TTest class to do a paired t-test, comparing the baseline rankings with the rankings produced by different configurations of the pipeline. The baseline system itself, as mentioned above, achieves an MRR of 0.4375. Just adding the stemming and case folding increases MRR to 0.5625, with a p-value of 0.05. Using TF-IDF in the cosine similarity measure further increases the MRR to 0.6, with a p-value of 0.15. Finally, using Jaccard as a similarity metric instead achieves an MRR of 0.5375, with a p-value of 0.08. Thus, one can see that the biggest improvement is achieved by just adding stemming and case folding. Using TF-IDF weighting further improves the rating, but the confidence value is lower. Jaccard does relatively worse compared to the other two improved versions.