Joseph Chee Chang (josephc1)

11-791 Software Engineering of IIS

October 21, 2014

Homework3

Task2 Components

MINIMAL STEMMER

String: Utils.minimalStem(text : String) - A minimal English plural stemmer based on org.apache.lucene.-analysis.en.EnglishMinimalStemmer.

PORTER STEMMER

String: Utils.porterStem(text : String) - A standard Porter stemmer based on org.tartarus.snowball.ext.-PorterStemmer.

STOPWORD LIST

Removes punctuation tokens and also filter each token using org.apache.lucene.analysis.StopAnalyz-er.ENGLISH_STOP_WORDS_SET.

SPACE TOKENIZER

List<String> : Utils.spaceTokenizer(doc : String) - A simple tokenizer that splits sentence by spaces.

STANFORD TOKENIZER

List<String> : Utils.stanfordTokenizer(doc : String) - A tokenizer based on the Stanford Core NLP Packages.

COSINE SIMILARITY RANKER

double : Similarity.computeCosineSimilarity(queryVector : Map<String, Double>, docVector : Map<String, Double>) - For two given sparse matrixes, normalize them to unit matrixes and compute the dot product.

TFIDF

double : Similarity.computeTfidfSimilarity(docVector : Map<String, Double>, Integer : queryId) - For a given document sparse matrix and its query Id, retrieve the DF for documents with the given query Id and compute the TFIDF weighted matrix as output. The IDF formula variant from Okapi is used for better performance. The output matrix can than be passed to Cosine Similarity Ranker to measure weighted similarity.

OKAPI BM25 RANKER

double : Similarity.computeOkapiBM25Score(queryVector : Map<String, Double>, docVector : Map<String, Double>, queryId : Integer, k : double, b : double) - For two given sparse matrixes, normalize them to unit matrixes and compute the Okapi BM25 score (k=1.2, b=0.75). Document length is omitted for better performance, since the size of the corpus is so small, its hard to get a good average.

Performance

Based on the components listed above, I manually tried different combination of token extraction / clean up and rankers. All changes lead to better MRR score comparing to the baseline. However, to insure statistical significance, I also calculated the p-value of different component combinations comparing to the baseline. The results below shows that adding stemming and stop word filtering does not yield significant improvement (p-value .168 > .1), but using the Stanford Tokenizer in combination does (p-value .036 < . 05). For different rankers, the TFIDF weighted version of cosine similarity ranker does not show statistical significant improvement (p-value .157 < 0.1), most likely due to the fact that the corpus for each query ID contains too few documents to yield reliable IDF values. On the other hand, the Okapi BM25 ranker yield significant improvement (p-value .029 < .05). The full system, using stemming, stop word filtering, Stanford Tokenizer with the Okapi BM25 ranker performed best with the MRR score of .7625. The runtime of different system does not show significant difference (all around to 1 second), so is not reported.

Systems	MRR	P-value to baseline
Cosine (Baseline)	0.4375	N/A
Baseline + stemming + stopwords	0.5167	0.168 > 0.1
Baseline + stemming + stopwords + Stanford tokr	0.5958	0.036 < 0.05
TFIDF Cosine	0.5125	0.157 > 0.1
TFIDF Cosine + stemming + stopwords	0.5208	0.165 > 0.1
TFIDF Cosine + stemming + stopwords + Stanford tokr	0.5708	0.062 < 0.1
BM25	0.6750	0.029 < 0.05
BM25 + stemming + stopwords	0.7250	0.006 < 0.05
BM25 + stemming + stopwords + Stanford tokr	**0.7625**	**0.001 < 0.05**

Baseline Error Analysis

Of the 20 queries, the baseline system only one of them correctly (queryId 10). Here we analyze the 19 errors by different categories.

QUESTION TYPE ERROR

Symptom: The retrieved document is relevant to the query and uses similar words, but is not answering the question.

Effected Queries: 1, 2 (im not sure the gold standard is correct), 3, 4, 12, 13, 14, 16, 18, 19,

Possible Solution: Classify the queries to predict the answer type, and filter out documents that does not contain the correct answer type.

Difficulty: High, as it would require a question type classifier and a knowledge base.

TOKENIZATION ERROR

Symptom: Sentences not tokenized into meaningful segments. For example, containing tailing punctuations, “China’s” should be separated into “China” and “’s", “U.S.” or “A.D.” should NOT be separated as they are one entity.

Effected Queries: all

Possible Solution: Using a better rule based or pre-trained tokenizer.

Difficulty: Low, as many past research as well as library packages (lucene, stanford) already exist.

CASING MATCH ERROR

Symptom: Query and document could be referring to the same thing but using different upper / lower casing, and thus the similarity metrics would see them as different dimensions.

Effected Queries: 5 (*sorrow*), 17 (*corn)*

Possible Solution: Use lowercase for all inputs.

Difficulty: Very Low.

FUNCTION WORD ERRORS

Symptom: Many function words and common words, like “*is”, “the”, or “that”*, is not informative when retrieving documents, but is treated with the same importance.

Effected Queries: all

Possible Solution: 1. Use a stop word list to filter out uninformative function words. 2. Use a similarity ranker that take this effect into account, like TFIDF Cosine Similarity or Okapi BM25.

Difficulty: 1. Low, as many predefined stop word lists are already available. 2. Moderate, requires implementing different ranker and a global context.

System Architecture and Design

Most architecture are as provided by the archetype,and design patterns used are the same as in previous homework. Some new classes as follows:

edu.cmu.lti.f14.hw3.hw3_josephc1.utils.Utils **-** A class with a private constructor and only static public functions that stores the different components mentioned in the first section of this report.

edu.cmu.lti.f14.hw3.hw3_josephc1.utils.Similarity **-** A class with a private constructor and only static public functions that ranks the documents using different strategies as mentioned in the first section of this report.

edu.cmu.lti.f14.hw3.hw3_josephc1.utils.MemoryStore - A Singleton Pattern class that stores global context for different query IDs, i.e., document lengths, document frequencies, number of documents. The key for retrieving different attributes are in the Utils class.