

CSC 583: Programming Project 1

Andrew Zupon

1 Code Description

- Make sure all wiki test docs are in `src/main/resources/wiki-subset-20140602` if you want to build an index.
- Unzip the indexes you want to use in `src/main/resources/`
- The indexes will only build if they don't already exist. To build the index, choose which kind you want ("stems", "lemmas", or "plain") on line 52 of `zupon_andrew_project.scala` and run `sbt run`. This will take between 1 and 2 hours depending on the index and hardware.
- To test the system using the "stems" index (the best performing one), run `sbt test`. This will print results for the naïve and improved models using both default and modified similarity functions. If the index doesn't already exist, this will try to build it!
- If you want to test the model with different indexes than "stems", you'll need to uncomment lines in `QueryEngineTest.scala` between approximately lines 169 and 212.

2 Indexing and Retrieval

This section describes how I built the various indexes used for retrieval, along with how I built the queries from the Jeopardy questions and categories.

First I will discuss how I preprocessed the Wikipedia documents. The first step was to gather the directory into a list of files, then loop through each file. Within each file, I joined the lines into one long string and then split on `\n\n[`, since each Wikipedia page title is enclosed in double brackets and follows the preceding page after two empty lines. This turned the file string into

an Array, where each entry in the Array was a Wikipedia page. Next, I looped over each entry in the array and performed various regex replacements to eliminate "junk" from the Wikipedia dump. This included hyperlinks as well as metadata about files, images, and references. From there, the processed text was funneled to the indexes.

I ended up building three different indexes: one that used stemming (`indexStems`), one that used lemmatization (`indexLemmas`), and one that used neither stemming nor lemmatization (`indexPlain`). For all three indexes, I split each line (which is an entire Wikipedia page and its title) on a dummy separator symbol that I inserted during preprocessing. The head of the split is the `pageTitle`, and will get written into the `'docid'` field with the Lucene writer. The tail of the split is the `pageText`. This text will ultimately get written into the `'text'` field, but how it is processed depends on the specific index.

For the stemming index, which uses the `StandardAnalyzer`, I simply write the entire `pageText` to the index; the `StandardAnalyzer` takes care of the rest. For the lemmatizing and plain indexes, which use the `WhitespaceAnalyzer`, I need to do additional processing on the text. To get lemmas, I use the `CoreNLP Processor` to tag and lemmatize each sentence of the text, extract the lemmas, and join them as a string with a single space character in between. For the plain index, which does neither stemming nor lemmatization, I also use the `CoreNLPPProcessor` to extract the individual tokens of each sentence before similarly joining them with spaces. In hindsight, I probably could have done this part more efficiently, but the way my code was structured up to this point funneled the "plain" text in this way.

For this initial naïve approach, beyond the regex filtering of hyperlinks and other metadata, I did not filter out any stop words from the text, nor did

I do any additional processing on the 'docid' field.

Next I will discuss how I build the query from the *Jeopardy!* question. Again, I started with some minor preprocessing. I split the file on empty lines, so each line contained the question category and the clue itself. I then simply concatenated all the words in the category and clue into a string and fed them to Lucene as an OR query. For the queries, there was a funneling step similar to when I built the index. Depending on if I wanted to search using stems, lemmas, or neither (plain), it will initialize the appropriate QueryEngine object and use the corresponding Analyzer (Standard or Whitespace) and direct it to the right index. Again, for this naïve approach I did not filter any stop words from the query, but I did delete the sequence (Alex: from the clue, which indicates a comment from the host Alex Trebek and does not actually contribute to the answer. Including Alex in the clue might negatively affect the results.

3 Measuring Performance

In this section I will discuss the initial results using my naïve retrieval approach. In the *Jeopardy!* gameshow, the goal is to get the unique right answer to the clue. Because of this, my measure of performance depends only on the top 1 result; if the correct answer is close to the top but not the right choice, it is considered incorrect. To capture this I use precision at 1 to measure the performance of my system. To calculate this, I simply take the number of correct results divided by the total number of questions (100 in this case). results for this initial retrieval model are given in Table 1. So far, we see that stemming performs the best, followed by lemmatization.

Index Type	Precision@1
Stems	0.18
Lemmas	0.16
Plain	0.15

Table 1: Naïve results with default scoring. Results are Precision at 1 out of 100 test sentences.

4 Changing the Scoring Function

Perhaps one reason for the lower scores is the scoring function. By default, Lucene uses BM25 (previously it used a vector-space scoring function based on *tf-idf* weighting). The two important parameters in BM25 equation are k_1 , which normalizes term frequency, and b , which normalizes document length. Lucene's default sets $k_1 = 1.2$ and $b = 0.75$.

Considering many of the Wikipedia pages provided here (and in general) are extremely short, sometimes only being redirects to other pages, I experimented with changing the value of b to change the scoring function. Setting $b = 0.1$ yielded improved results compared to the naïve approach, as shown in Table 2.

Index Type	Precision@1, $b = 0.1$
Stems	0.39
Lemmas	0.21
Plain	0.21

Table 2: Results with modified BM25 scoring. Results are Precision at 1 out of 100 test sentences.

By setting $b = 0.1$, I penalize longer pages less. Considering the short Wikipedia pages are often stubs, it seems more likely that a longer page will be more relevant for a *Jeopardy!* answer than a shorter page. This is just a hypothesis for now.

5 Error Analysis

6 Improving Retrieval

Query: \texttt{Not to be confused with karma, krama is a popular accessory sold in cambodia; the word means "scarf" in this national language of Cambodia CAMBODIAN HISTORY \& CULTURE}

Correct answer: \texttt{khmer language}

Using modified BM25 Similarity:

Hit 1:	DocName: History of Cambodia	DocScore: 59.53534698486328
Hit 2:	DocName: Politics of Cambodia	DocScore: 59.104557037353516
Hit 3:	DocName: Khmer language	DocScore: 58.31053161621094
Hit 4:	DocName: Khmer Rouge	DocScore: 57.4572868347168
Hit 5:	DocName: Economy of Cambodia	DocScore: 51.73160171508789
Hit 6:	DocName: Siem Reap	DocScore: 48.53266525268555
Hit 7:	DocName: Ho Chi Minh City	DocScore: 48.411109924316406
Hit 8:	DocName: Ikat	DocScore: 47.433082580566406
Hit 9:	DocName: Paan	DocScore: 46.23930358886719
Hit 10:	DocName: Mekong	DocScore: 46.0582275390625

Query: Don Knotts took over from Norman Fell as the resident landlord on this sitcom THE RESIDENTS

Correct answer: three's company

Using modified BM25 Similarity:

Hit 1:	DocName: Don Knotts	DocScore: 61.50598907470703
Hit 2:	DocName: Three's Company	DocScore: 44.23488235473633
Hit 3:	DocName: Tim Conway	DocScore: 35.788169860839844
Hit 4:	DocName: John Ritter	DocScore: 33.25556564331055
Hit 5:	DocName: Bill Bixby	DocScore: 30.387065887451172
Hit 6:	DocName: Matlock (TV series)	DocScore: 30.287059783935547
Hit 7:	DocName: San Pedro, Laguna	DocScore: 29.240985870361328
Hit 8:	DocName: That '70s Show	DocScore: 28.738615036010742
Hit 9:	DocName: Malta	DocScore: 28.33993148803711
Hit 10:	DocName: County Mayo	DocScore: 28.2568359375

Table 3: Examples where the highest-rated answer contains a word or words present in the clue. *Jeopardy!* answers are unlikely to contain words that are in the clue (excepting punny categories), so skipping results that *do* contain overlapping words might fix these errors.

Query: 1980: "Rock With You" '80s NO.1 HITMAKERS
Correct answer: michael jackson

Using modified BM25 Similarity:

Hit 1:	DocName: Orchestral Manoeuvres in the Dark	DocScore: 27.115528106689453
Hit 2:	DocName: Cold Chisel	DocScore: 25.40660858154297
Hit 3:	DocName: Wolfman Jack	DocScore: 25.255949020385742
Hit 4:	DocName: The Runaways	DocScore: 25.107824325561523
Hit 5:	DocName: Helix (band)	DocScore: 24.45211410522461
Hit 6:	DocName: Whitesnake	DocScore: 24.35768699645996
Hit 7:	DocName: Duran Duran	DocScore: 24.109310150146484
Hit 8:	DocName: Laibach (band)	DocScore: 24.034826278686523
Hit 9:	DocName: Glam metal	DocScore: 23.971431732177734
Hit 10:	DocName: The Moody Blues	DocScore: 23.339218139648438

Query: 1989: "Miss You Much" '80s NO.1 HITMAKERS
Correct answer: janet jackson

Using modified BM25 Similarity:

Hit 1:	DocName: Helix (band)	DocScore: 27.506011962890625
Hit 2:	DocName: Neneh Cherry	DocScore: 25.372238159179688
Hit 3:	DocName: George Michael	DocScore: 25.11778450012207
Hit 4:	DocName: I Love the '80s Strikes Back	DocScore: 25.046329498291016
Hit 5:	DocName: Book of Love (band)	DocScore: 23.877687454223633
Hit 6:	DocName: Gloria Estefan	DocScore: 23.783161163330078
Hit 7:	DocName: Halle Berry	DocScore: 23.55036163330078
Hit 8:	DocName: Annie (musical)	DocScore: 22.938392639160156
Hit 9:	DocName: Orchestral Manoeuvres in the Dark	DocScore: 22.803762435913086
Hit 10:	DocName: NBC	DocScore: 22.757150650024414

Table 4: Examples of where the words in the clue and category are not informative enough to get the right answer. Could possible be solved with phrase queries using the song title given in the clue.