IBM Watson Report
CSC483 Final Project

Professor: Mihai Surdeanu
Author: Trey Bryant
Fall 2020

Important Note: Link to Github Repo - https://github.com/tr3ylbry/IBMWatson

- See README for index file links

Section 1: Indexing & Retrieval

- *Describe how you prepared the terms for indexing?*

  Preparing the retrieved terms and components of the wiki documents for indexing varied depending on the indexing method chosen from the start menu. However, regardless of the indexing method chosen, the wiki pages were prepared in the same way, in that there were certain formatting issues that needed to be addressed.

  1. Titles are surrounded by brackets like so - `[[Wiki Page Title]]`
  2. Categories are indicated like so – `CATEGORY: I'm a category`
  3. Headers were ensconced in equals signs similar to – `==Header==`
  4. Various parts of text were surrounded in these tags – `[tpl]content inside[/tpl]`

- *What issues specific to Wikipedia content did you discover?*

  Due to the issues detailed above, I needed several 'parse' methods that parsed out the brackets ('[') and equals signs ('=') and such so that I could isolate whatever component was obstructed by the extraneous symbols. I then added these components to a Document object, with each representing a Wiki page. I did this by calling my addPage() method every time I encountered a new title, which would then add to the Document object all the categories, headers, and lines of text that had accrued in their respective variables. By some inspection/debugging, I found that there were occasionally some lines that were classified as titles that may or may not have contained titles, but were otherwise a long line that happened to start with a bracket. In this case, I attempted to constrict my parameters for what constituted a title in an attempt to address this, but also there are so many wiki pages, and many edge cases, so it's difficult to know how well it worked, but my scores aren't so abnormal.

- *Describe how you built the query from the clue?*

  To build the query, I simply concatenated both the category and the clue, and since added the category to the document as part of two different fields (category itself, and also 'body'), it seems plausible that it should be able to be found by the searcher. I did not experiment with using a subset of the words in the clue, I opted for a 'the more the better' approach, but if I was to use a subset, I would get rid of stop words as a first move, and potentially try to pare things down further from there.

Section 2: Measuring Performance

In this case, I only opted to use P@1 precision, due to both time constraints and ease of implementation; for the top document at position 1 that has the highest relevancy score, it stands to reason that this document might be the correct answer and in a fair amount of cases, this holds true. I calculated the score using this method by simply keeping a count of questions answered correctly by the top document and also by the total number of questions. The precision was then count / total. In addition, this metric provides a very uninhibited view of the true performance of each combination of index and query method, and thus seemed like the best choice.

Section 3: Changing the Scoring Function

From the opening menu there is first the option given to run the program with different indexing types:

1. None (neither lemmatization nor stemming)
2. Lemmatization
3. Stemming

After this, you are given the option to pick a query similarity, out of these four:

1. BM25 (the default)
2. Boolean
3. TF-IDF Weighting
4. Jelinek-Mercer

The table presented below are the achieved scores with each combination of index and query method. One can see that using no special indexing with the Jelinek-Mercer query method yielded the best results. You will also observe that for my lemmatized indices, the data is unavailable across the board. For reasons I was unable to discern, there was a bug that only affected this index type. I looked extremely hard but was not able to discover the reason prior to the deadline, but I was informed that lemmatized scores could be expected to hold similar to the None type.

	BM25	**Boolean**	**TF-IDF**	**Jelinek-Mercer**
None	0.20	0.09	0.0	0.27
Lemmatization	---	---	---	---
Stemming	0.21	0.12	0.0	0.25

Section 4: Error Analysis

- *How many questions were answered correctly/incorrectly?*

  As mentioned above, we can see that the None type indexing method paired with the Jelinek-Mercer query method was the best system used (barring the fact that I was unable to successfully debug my lemmatizer). Since there were 100 questions in 'questions.txt', we can see that this system answered 27 correctly which is a failing grade by far if we consider a typical grading system, but perhaps not bad at all by a computer that is parsing through hundreds of thousands of Wikipedia pages, in order to answer questions that are intended for humans.

- *Why do you think the correct questions can be answered by such a simple system?*

  The phrase 'simple system' is relative I suppose, there were a lot of different components leading up to retrieving an answer: the parsing of the wiki documents, the indexing method, the parsing of the questions/clues, transforming them into a query, and then retrieving a list of documents relevant to the query. To more accurately answer the question, this system works on a base level by matching the words in the query (that is constructed by the question and clue given) to the information in the documents that have been created by whatever method has been used to index the wiki pages. At a very core concept level, the program works by word-matching, whereas a human would have to remember this type of information. However, in a way, I suppose a human playing Jeopardy would be doing word-matching to match the question they were given to information they had stored previously by reading. So in a way, the process is similar.

- *What problems do you observe for the questions answered incorrectly?*

  I think a big issue for the incorrectly answered questions could be the prevalence of stop words. There exists no part of the current implementation that addresses stop words so as to eliminate or otherwise mitigate them prior to retrieving an answer. As mentioned above, a sensible next step in making this answering machine more powerful would be to address this issue.

- *What is the impact of stemming and lemmatization on your system?*

  Unfortunately, without the functionality of my lemmatizer, it is hard to very accurately answer this question; upon observation of the table provided in **Section 3**, one can see that the scores using Stemming to Index are slightly higher than those using the None-type method, *except* for when Jelinek-Mercer is used as the query method. This combination with None-type indexing and Jelinek-Mercer yielded, as previously stated, the highest score out 100 questions asked. Given the inconsistent performances between query methods, the jury is still out on the best configuration considering indexing alone. Lemmatization might beat them both, but I will have to do some further debugging to resolve this issue.