THE WATSON PROJECT

VOL EST 2000 New York February 2011

Core implementation

Please use the following to compile our code on the command line: java index.java
Otherwise, please follow directions on our README for running the code.
Our main idea for the project was to use the category of the jeopardy clue with the category tabs of each wiki document as part of the query. Each document in our index has content, a category, and a title. To prepare each document for indexing its content and categories are downcased, tokenized, normalized, and stemmed. This also removed all of "Alex's hints." Stop words are also removed. This is done with an English Analyzer. The same process was done to the query. We used the entire clue to find matches within the content or category of the wiki document rather than a subset of it.

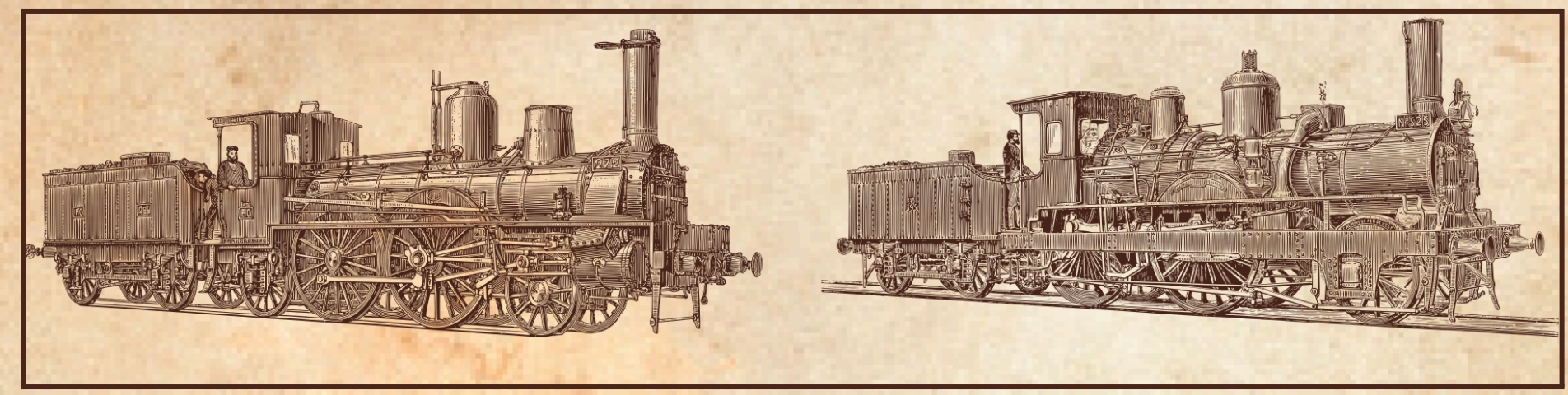
The category of the clue was used to match with the categories of the parsed documents by concatenating the category and clue and searching into the content of the page. An issue we found during this is that the articles often contained embedded links. To remove this we added in the tokenization process with removed any non alphanumeric characters. Another issue with the wiki content is the title brackets. many other lines contained the double brackets "[[" ,which prompted us to have to search for ending double brackets to denote a title. After the preprocessing was completed, we decided to concatenate the categories with the content to perform the search. This yielded an overall percentage boost for precision.

Measuring Performance

To measure performance we mainly used P@10 and MRR. Our P@1 remained a steady 22% throughout testing. Our P@10 is 47% and our MRR is 29.41% . We believe these are accurate measurements of our program .P@10 is measuring how many times the correct jeopardy answer is in the top 10 hits. We believe this is a good measurement because it shows that our ranking system is getting the documents in the higher ranks even if they are not in the #1 slot. The MRR takes in account the fact that the correct jeopardy answer is more often than not in our #2-#6 range meaning we are ranking them near correctly.

Error analysis

One of the main errors we found was with our category search. Many documents shared the same categories causing false hits. An example of this is for query 1 searching for "The Washington Post" with the category "Newspapers". The query for categories had many false positives for other news paper websites. One huge error we found was that we are getting a lot of hits in the top 10 that more closely matched the clue than the actual answer. An example of this is that for the answer "Michael Jackson" or "Janet Jackson", the top ten hits would feature 80's or 1980 in the titles. The clue for Michael Jackson was "1983: Beat It" and the category was "'80's No. 1 HITMAKERS." Another example of this was getting multiple results with "Jell-o" or "Jelly" in the title for the answer Kraft Food as that was the only clue. The matched documents were matching the clue rather than the correct answer. Overall this gave us around 22% accuracy for 100 questions posed. Stemming is a large reason why we get correct answers at rank 1 as it gives us the ability to match plurals whereas previously we had not (see: newspapers vs newspaper). Another reason we are able to get this accuracy is a later implementation in which we noticed some answers have multiple options and we were not accounting for that. Once we added in parsing for multiple answers there was a noticeable increase in our accurate hits. We think that such a simple system is able to answer these questions due to the combinations of our implementations. Indexing the documents helps organize the data, the similarity scoring that ranks the top documents, stop words and stemming are done to help matches be found. Stemming helps to increase the words a match can match too by chopping off useless plurals.



Tries and Fails Our Defense

One of the reasons that we think our project is worth an A is because of the amount of effort we put into trying new ideas for querying. While not all of our ideas worked for our performance we still learned a lot about why our project was failing; we think this is the most valuable portion of this experience.

One of our first tries was to try separating the category and clue parsing by parsing the clue into the content of the document and the question category into the categories section of the wiki docs. This brought us a much better performance than what we had at the time. now we know that concatenating the clue and category. When we had them split we tried to add weighting to the clue or categories. This was a large decrease in performance. We are assuming this is based on the fact that any one of the clues and categories was not more important than another. boosting categories inflated other documents because many documents have the same categories.
boosting the clue with weights most likely did not help because certain words in the clue were matching and over inflating the irrelevant words in the clue. Another implementation we tried was querying the categories into the top 10 frequent words of a wiki document. This was our second highest performance, it unfortunately was bested by concatenating the clue and category. We believe this didn't work because many of the frequent terms were not quality terms. If we could filter the frequent terms more we believe this could improve its performance.

Another try and fail is Synonym Mapping. while some words did not match with stemming, some of them may have been a synonym of the category. Unfortunately there were errors in getting this to run on our systems but the idea was great in theory and if combined with term frequencies could possibly be better than our current system.