Question Answering Part 1

Evan King, Abhi Gupta, Alex Sommer, Alex Pomerank

esk79, aag245, ajs455, arp257

November 3, 2016

I. Baseline System

Our system searches through a given question and parses it for the question type (who, when where). It then removes words in the question text that appear in a set of "50 most common English words" including "the", "a", etc. This leaves us with keywords from a question for us to use in searching through our documents. For example, the question "What is the capital of Ohio" would return ("What", ["capital", "Ohio"]).

We then parse through a given directory and transform each file into a list of words. This provides us with a data set to search through. We take our question keywords and put them into a set. We parse over our entire directory of words, and if the word we are parsing is in the set of keywords, we note the index of this word in the array.

Once we have parsed the directory and returned a list of indices, given an arbitrary window size, we return a list of index ranges which have this said window size. The window size is defined to be the inclusive total number of elements that fall in those indices. The list is returned in decreasing sorted order by our scoring metric, density, which is simply the number of indices found in a range divided by window size. For example, if I was given a window size of 3, and an index array of [1, 3, 4, 5, 12], the function would return [(1, 3), (0, 1), (1, 2), (0, 0), (4, 4)] . The latter list marks the index ranges sorted in decreasing order of density. For instance, the range (1,3) would receive a density score of 3 because it contains 3,4, and 5, and also fits within the designated window size of 3.

II. Justification

The underlying assumption of our system is that the answer to a given question will –most likely– include a re-arranged form of tokens that appear in the question text. For instance, if the question were "Who killed Abraham Lincoln?", one could expect that the answer would probably reside in text that includes the tokens "Abraham", "Lincoln", and "killed" in a somewhat local distribution. Consider the possible phrases appearing in the ranked document corpus such as "John Wilkes Booth killed Abraham Lincoln" or "Booth killed Lincoln in 1865". The property that both these phrases share is that they at least explicitly make use of "killed" and "Lincoln", while the other subject ("Booth") can be inferred as the subject of the action "killed".

Although this assumption is quite lenient, it leverages a coherent assumption (namely, colocation and density of keyword tokens) about answers to basic factoid questions in this assignment.

III. Analysis

Mean reciprocal rank of our system over 182 questions is 0.052, and 168 questions had no answers found in top 5 responses. This shows that our system is not performing as well as we expected, which could be for a series of reasons. The most likely reason we deduce is that our system finds several occurrences of the maximum keyword density in the directory, however when returning the list of these densities to other parts of the system, they are returned in decreasing order and recency. By nature of our implementation,

two equal densities may be returned, but the density found first would be returned first. For example, if our system finds 100 instances of the highest density in the directory, but the first 5 instances occur in the first file of a directory, those would be selected as our answers. This does not tend to be a good way to handle ties, as one given document may not be relevant to the correct answer. Perhaps spreading our answers across several files, or at least preventing answers from all being concentrated in one file, would have increased our score. Furthermore, we speculate we may want to use our baseline system as our initial pre-processing mechanism, and then with given sets of our highest densities, determine word relevancy to the question somehow. Nevertheless, this analysis on our system provides us useful feedback on potential improvements for our final system.

IV. Final System Proposal

We intend on creating a final system that will be an iteration of our baseline. We have identified a series of aspects that need to be adjusted and improved upon in order to build a comprehensive question answering system. To begin, our text data is riddled with unnecessary metadata. By cleaning this data we will have a less obstructed document and be able to more correctly determine answers.

Beyond this, and more importantly, we plan to build on our assumption that that the answer will be found close to words that are in the question. Currently, our clustering is one dimensional and we use a fixed window size. By improving on our clustering we will be able to improve our passage retrieval. We have identified that not all words in the question are directly found (or relevant in the context of the question) in the document. We plan to use Word2Vec, a module which allows for vector arithmetic on words. This will give us the ability to, along with many other things, compare two words and determine a "similarity percentage". With this, we can mark not only exact words that were found in the question, but words that are similar (given a specific threshold). This will increase the number of nodes we'll be clustering on and hopefully improve passage retrieval. We would also like to make our cluster more than one dimension if possible, although the additional dimensions we would cluster on are still up for debate.

V. Work Breakdown

1. Parse all questions into keywords: Alex P
2. Read related files for a question ID into a string of words: Abhi
3. Find all instances of the set of keywords in the string of words: Abhi
4. Cluster the found indices of the keywords into a list of cluster locations: Alex S
5. Find the answer for all the cluster locations using the API: Evan
6. Write the answers to an output file: Alex P
7. Combine different parts: Alex P and Alex S
8. Write report: All