Question Answering Part 1

Evan King, Abhi Gupta, Alex Sommer, Alex Pomerank

esk79, aag245, ajs455, arp257

November 14, 2016

I. QA System Approach and Description

For this assignment, we hypothesized that due to the nature of the short and direct question/answer pairs in the corpora, a form of bag-of-words and chunking would be most effective in tackling the problem. From a high level, we prune down the question text in an attempt to only extract the essential keywords to that question, search all corresponding documents for a given density of those keywords, and then employ a semantic tagger at the end to extract our answers.

Our system essentially makes the assumption 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."

In addition, we've added a bunch of new features that seek to contextualize some of our assumptions and eliminate false positives in our guesses. First, we improved both our density calculation and chunking logic. Instead, we now define a gap between keywords from the question that appear in the corpus, and then compute our density based off of the count of all keyword tokens that appear within that gap range. In addition, once we have sorted these by decreasing count value, we adjusted our chunking logic to expand to find a sentence end in both the trailing and head ends of the chunk. This is a worthwhile assumption to make, we found, as sometimes the actual answer is located out of this exact density range that we found when looking for keywords. Furthermore, it is usually a relatively small expansion to the whole chunk, and sentences generally contain logically or semantically related ideas.

Finally, we discovered one specific preprocessing assumption we could make about tokens in the search corpus that helped find related orthographic and syntactic forms without much work, mostly by treating each token as a set of characters. The motivation behind this adjustment was to catch similar word forms in the text such as: "Booth decided to kill Lincoln in Ford's Theater." In this case, we would have started by initially looking for "killed" as a keyword in the corpus (because it is derived directly from the question), but in this case we are able to pick up "kill" as a keyword, as well, because 4/6 characters match in the string.

We considered using synonym and semantic comparisons models to perform a similar sort of token constraint relaxation, however, we found that it ran far too slowly to perform such checking on every word, and since the approach is only really effective if consistently applied to all words, we decided not to include this relaxation in our final system. Similarly, the synonym-based approach did not yield what we hoped for exactly either. It ultimately relaxed too many constraints on similarity, and we got far too many low-quality answers to the

questions.

II. External API's

Our system relied on the TextAnalysis API to help us simplify several steps based on information the API could provide us. Specifically this API extracted the answer to questions from our final chunks using Named Entity Recognition and matching this with the question type.The system also inferred what the answer type should be, such as Who = PERSON, When = DATE, etc. This system effectively allowed us to use relevant information to deduce what we were looking for, and then after we processed and selected the most likely chunks, pull the correct information without us having to worry about many specifics.

III. Comparison to Baselines

Our baseline system was a rudimentary approach in comparison to our final system. We had initially implemented a system that took a local density per window size to determine relevancy. This means, for example, if the question was "Where is Belize located?", the tokens would be "Belize, located". With the window size set to 10, the system would find all indices of these keywords in our documents, and cluster them into sets of 10 words. Our "density" was computed as the frequency of the words over the window size, and these clusters were sorted in descending order based on this density. As a result, our system would look at the clusters that had the highest number of keywords, and select those clusters as most likely to have the correct answer. As mentioned in our initial report, the issue with this implementation is the assumption that the distance between keywords in a specific cluster does not matter. For instance, a 10 word cluster with keywords at indices 2, 3, and 4 may not be weighted higher than a response with keywords at indices 1, 5, and 9. Holistically speaking, a higher concentration of keywords may indeed signify a better answer than keywords scattered around. Furthermore, the issue remained that the system does not scatter its selected answers across documents. If 30 clusters had the highest possible density, and the first 5 occur in the same document, our system would select those 5. If the document it selects from is completely unrelated to the topic, we have essentially placed all our guesses in one document. We speculate that "diversifying our portfolio", or spreading guesses across documents, would yield better results. We furthermore recognize that gaps between keywords impact the likelihood of a phrase's validity, and thus use this intuition in our new, final system. There are without a doubt better results with this consideration, and overall we gained significant insights through our baseline system.

IV. Question Walkthrough Description

In order to answer a given question we first break up the sentence into a list of tokens. We then take the first word as the question type (who what or when) and the rest of the words as corresponding keywords. Then, we filter out commonly used words, like "is", "the", "and," etc.. Next, we take the relevant directory for this question, and read every word into an array of words, word list. We then go through the word list, and store the indices of the words that are also in the keywords. Next, we cluster the indices together that have the highest number of matches keywords, generating a list of tuples of indices where there are the highest density of keywords together. We then look at each one of these tuple ranges, and look back into the word list to get the actual words that correspond to the question, cluster words. We filter out commonly used words again, and then use the Stanford API to get the named entities in the string. For those that match the question word (e.g. Who must be a person), we find the document that it was found in, using the tuple of start and end indices, finding the corresponding document ID. We then go through these, and write to the answer file the document ID and the answer, at a max of 5 times. There are also various small

operations and processing steps that are not included in this description but clearly evident in the relevant code.

V. Question Walkthrough Code Substeps

1. question: where is montenegro
2. question type: where
3. raw keywords: [is, montenegro]
4. keywords: [montenegro]
5. all tuples: [(11503, 11525), (22040, 22081), (47179, 47196), (9872, 9892), (15203, 15223), (37997, 38055), (52223, 52255), (24512, 24558), (56404, 56430), (1032, 1081), (6040, 6045), (31, 31), (82, 118), (1448, 1473), (1576, 1587), (13157, 13164), (14420, 14437), (14938, 14956), (20329, 20345), (21345, 21377), (52802, 52832)]
6. first five: [(11503, 11525), (22040, 22081), (47179, 47196), (9872, 9892), (15203, 15223)]
7. raw text for first five:
   (a) text: Dr. Vojislav Seselj asked the members of the Serbian Radical Party [SRS] and its sympathizers, who gathered to hail their leader on
   (b) text: While talking to several senior Croatian Navy officials, we agreed that Montenegro (when it truly is Montenegro) will have the legitimate right to establish its own navy, in accordance with its defense needs, and also to take all passive and active
   (c) text: It has opened the way for very dangerous developments in Montenegro,' says Mr Dragoslav Micunovic, a democratic
   (d) text: I am working to help Montenegro overcome this difficult and tragic crisis of the collapse of former Yugoslavia as painlessly
   (e) text: [Perovic] The idea of union with Herzegovina has the additional goal of destroying any thought of any Montenegrin position in
8. text with word that matches the question type: I am working to help Montenegro overcome this difficult and tragic crisis of the collapse of former Yugoslavia as painlessly
9. matching word and index: yugoslavia, index 9882
10. file: 23

VI. Analysis

Our original system scored a mean reciprocal rank score of 0.052. Our new, improved system scored 0.123, showing an increase of more than double in score.

Our new system's considerations, specifically our reliance on the gaps between clusters to signal relevancy as an answer, helped alleviate many problems we ran into with our baseline model. We see that the issue of highest density clusters appearing first in the initial set of documents is not necessarily prevalent in our new system. As more relevant answers appear in question keyword clusters, we find ourselves identifying more correct phrases, and as a result our system produces better answers.

We are no longer picking only answers from the first section of documents, and rather sift through more documents and find more accurate clusters. Our MRR score doubles most likely due to the fact that our answers are no longer concentrated in one document, and therefore produce results from different documents that may end up focusing on different topics.

Our strongest component of the system is the assumption that the keyword questions are always going to be near the answer. As a result, we can pinpoint exactly where we want to look (as seen through our density and gap calculations), and select text accordingly. This is a very simple and straightforward method,

and as a result we believe that we have a ceiling on how accurate this system can be. The weakest aspect of our system has to be the reliance on the exact keywords given: as previously mentioned, using synonyms and similar roots would have been the best option as we could expand the capabilities of our system. For instance, if I ask "Who invented the first television?", I could use synonyms to deduce that first could be replaced with original, or prototype, and thus we could sift through the documents as if these were also keywords. Similarly, we believe that roots, or for instance taking any word that had invent in it (such as inventor, or invent, or invention) would all direct us to the correct answer, and increase our scores.

VII. 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, Evan K
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