CS 4740 Project 3 report

Saerom Choi (sc2233), Gi Yoon Han (gh336), Rui Liang (rl522), Xiaoyun Quan (xq44)

November 13, 2016

0 The task and our strategy

In this project, we aim to develop a Question-Answering system to produce answers to questions in `question.txt`. We divided this task into 4 parts:

- Query formulation: obtain the keyword query from the question that is suitable as input to our Information Retrieval system;
- Question classification: classify the question by its expected answer type that specifies the kind of entity that would constitute the answer;
- Passage retrieval: filter out passages from the provided documents retrieved by the Smart IR system
- Answer processing: extract a specific answer from the retrieved passages

Question Formulation and classification make sure that the question is well-understood, and then the query is passed down to "Passage Retrieval" part to make sure we only keep relevant sentences for further processing. At last, Answer processing will make sure that the answers are appropriately extracted and sorted accordingly.

We eventually finalized our system after running 14 experiments for different methods (See Section 3).

Based on the final system, we also provided a detailed analysis on the output answers, see Section 4.

0.1 Instruction to run the code

final.py has our final implementation. baseline_grammar.py is the code we used when experimenting with syntax and dependency parser. semantic.py is the code we used to experiment with semantic role labelling. final.py requires nltk and fuzzywuzzy libraries to run. baseline_grammar.py and semantic.py requires, in addition to nltk and fuzzywuzzy, spacy(spacy.io) and practnlptools(https://github.com/biplab-iitb/practNLPTools) to run. Have all the included sources in the same folder. To run on the development set, include `question.txt` from `doc_dev` folder on the same folder. If running the test documents, include the question and the docs, and some changes to path settings in the files are needed. They are near the top of each file. Change d_path to where the documents folder is located.

1 Baseline system

To start with, we built a baseline system with a simple method for each part of the task.

1.1 Query formulation We tried to extract the list of keywords by a few steps:

1) leave out the question word
2) remove stop-words by utilizing `nltk` package in python: the stopwords we used here are the `english` subset of the corpus of stopwords from `nltk.corpus`
3) remove punctuations from the question: the list of punctuations consists of ‘.’, ‘,’, ‘ “’,” ’“, ‘?’, ‘!’, ‘:’, ‘;’, ‘(’, ‘)’, ‘[’, ’]’, ‘{’, and ‘}’

Note that since the top 100 relevant documents have already been retrieved and are provided to us, the list of keywords is no longer needed to be passed to IR for document retrieval.

1.2 Question classificaiton Since the questions in `question.txt` only involves "where" "who" and "when" questions, we decided to classify the questions simply by the first word in it for our baseline system.

1.3 Passage Retrieval After having obtained answer types from the previous step, we can now retrieve passages from provided documents. We tagged each document with NER tagging from `nltk`, and filtered out those sentences that don't contain the desired NER tags according to the answer type. Here we specify NER tags to each answer type as follows:

- Who-type: PERSON
- Where-type: LOCATION, FACILITY, GPE
- When-type: TIME, DATE

The details (documentation, coding, etc.) for NER tagging can be found at https://pythonprogramming.net/named-entity-recognition-nltk-tutorial/

1.4 Answer processing For our baseline system, we decided to return the exact entity that matches with our answer types.

1.5 Baseline wrap-up While being as simple as possible, our baseline system is still expected to return some answers that make sense to the question, because the NER tagging is able to spot the matching entity since the documents retrieved are already ranked by its relevance to the question content, although the chances are not big. For example, our first answer to question 90 is exactly as expected in `pattern.txt` (Johan Vaaler). This is because the first PERSON tag appears in the 2nd document, and that corresponding entity turns out to be the right answer.

Not surprisingly, the mean reciprocal rank for our baseline system over 232 questions is 0.084. Amongst all questions, 204 questions had no answers found in top 5 responses.

2 The final system and its development

We ran 18 experiments in total to compare performances of different systems based on MRR score. Our best system reached MRR = 0.299 and has the following features that are different from the baseline system:

- Sentences that does not contain any keyword will be filtered out, so if a sentence contains at least one keyword, it will be kept;
- The passages retrieved are of 2 consecutive sentences, so if a sentence is to be kept (we call it a leading sentence), its following sentence will also be kept
- We limit the size of retrieved passages to 30, so as long as we retrieve 30 passages, we stop retrieving
- All the retrieved leading sentences will be ranked on Levenshtein distance by utilizing the function `partial_ratio` from `fuzzywuzzy` package
- Phrases that match the desired NER taggings will then be extracted as answers
- Answers will be ranked based on its average distance to keywords
- We used another time tagging for WHEN-type question, since the one from nltk NER tagging is not functioning

The following subsections are the series of experiments that led us to the final system.

2.1 Experiment 1: filtering sentences by keywords + sorting surviving sentences In the baseline system, we filtered sentences only by seeing if it contains any word with the desired NER tagging. This is over-simplistic, so the first experiment we conducted will be to filter out those sentences that is irrelevant to the question, i.e. it will be filtered out if it does not contain any keyword. After this, we also sorted the surviving sentences by Levenshtein distance via calling the `partial_ratio` function from `fuzzywuzzy` package (found at https://github.com/seatgeek/fuzzywuzzy). The MRR increased to 0.123 after implementation.

2.2 Experiment 2: keyword stem matching We then realized that it makes more sense to keep sentences if it contains any words with the same **word stem** as one of the keywords, for example, Question 89 asks "Where is Belize located?", then "Belize" will be one of the keywords here, and we would like to keep a sentence if it contains "Belizean" or any other words sharing the same stem. Note that we utilized the WordNet lexical database to find the stem and matching words. Upon implementation, we saw an improvement that MRR = 0.141. So this strategy will be retained for future experiments.

2.3 Experiment 3: adding time tagging for WHEN-type question Since the original NER tagging for `time` and `date` from `nltk` is not functioning well, we end up with using another tagging system found at https://github.com/nltk/nltk_contrib/blob/master/nltk_contrib/timex.py. For example, for question 103 "When did the vesuvius last erupt?" the expected answer is 1944 and we are able to catch it as our first answer on the list. But it is not flawless, unfortunately, that it would not be able to capture a few time-indicators such as '7th century' and 'December 26', and that is why it will mess up with question 185 "When is Boxing Day?" where the expected answers could be "ten days January 4", "26 December", "December 26" or "day(s) after Christmas". But this is supposed to be a minor case since we found that most of the answers to the WHEN-type will be years which will be captured perfectly with this tool. Upon implementation, the system increased its MRR to 0.162.

2.4 Experiment 4: removing the limit on number of answers extracted from each sentence From Experiment 2, we wondered if keeping multiple answer candidates in each sentence would improve performance. Recall that in our baseline system, we only keep the one answer from each retrieved sentence. Now we tried to remove that limit and kept all possible answers. We saw MRR rocketing to 0.258. This implies that we might have missed a lot of relevant information by limiting the number of answers only one per sentence. This experiment was done independently from experiment 3, so it was still without time tagging for WHEN-type questions.

2.5 Experiment 5: considering synonyms for keywords We wondered if it will capture more relevant information by adding synonyms for keywords to the list of keywords. We decided to utilize WordNet, again, for obtaining synonyms, and implemented it on top of Experiment 4. All the lemmas sharing same synset as a keyword will then be added to the list of keywords. But the MRR decreased to 0.22 upon implementation. This is probably an indication of redundancy that too many irrelevant answers were introduced. We decided to remove this feature for the following tests.

2.6 Experiment 6: Combining Experiment 4 with time tagging from Experiment 3 We then merged the implementation from Experiment 4 with the time tagging done in Experiment 3. MRR rose to 0.295 upon the merge.

2.7 Experiment 7: considering syntax similarity for sentence ranking Previously, we sorted the retrieved sentences only by comparing the sentence and the keywords, where grammar was not considered. We tried to incorporate syntax similarity in the hopes of improving performance. This experiment was based the implementation from Experiment 6, so we hoped it would outperform 0.295 Experiment 6 was producing. Syntax similarity was calculated by comparing the similarity of dependency parsing results between the question and a candidate answer sentence. For dependency parsing, we used Spacy(https://spacy.io/) for the implementation. For example, for the question “Where is Belize located ?” the resulting parse would be “advmod auxpass nsubj null punct”. We first retrieved with the same methods described above, and tried to replace ranking by keyword similarity with ranking by syntax similarity. The similarity score was evaluated using the same Fuzzywuzzy library. The result was meager 0.174. We tested if the syntax ranking was any better than no ranking at all. No ranking version of Experiment 6 resulted in MRR of 0.147, so it was better than having no ranking scheme, but it was significantly worse than keyword ranking.

2.8 Experiment 8: combining syntax similarity (90%) and keywords similarity (10%) for sentence ranking Then, we tried some combinations of syntax similarity and keyword similarity to see whether these would outperform keywords-only rankings. Combining 90% of syntax similarity score and 10% of keyword similarity score resulted in MRR of 0.205.

2.9 Experiment 9: combining syntax similarity (50%) and keywords similarity (50%) Taking 50 percent of each scores resulted in MRR of 0.267.

2.10 Experiment 10: combining syntax similarity (10%) and keywords similarity (90%) This experiment resulted in MRR of 0.278. After this experiment, we concluded that utilizing our method of utilizing syntactic similarity does not help with performance. Using only keyword similarity score proved to be superior. We suspect that this is due to the fact that our document sources were not vast and (only 100 documents as provided) there was only a small number of sentences that actually contained the answers, and thus keyword ranking successfully ranked the answers.

2.11 Experiment 11: Considering passive/active voice For the next experiment, still based on the implementation from Experiment 6, we tested to see if considering passive/active voice was helpful. This was done also by looking at dependency parsing results. Our methodology was that if the question contained an nsubj (nominal subject), it was classified as active. If the question contained an nsubjpass (passive nominal subject), it was considered passive. After the retrieved sentences were ranked by keyword similarity, the candidate sentences were examined for their passive/active voice. Sentences were prioritized if they were in the same voice as the question. For example, if the answer candidate sentences were ranked (1, 2, 3, 4) and 2 and 4 had the same voice as the question, it was reranked so that the order was now (2, 4, 1, 3). This experiment also performed worse than Experiment 6, resulting in MRR score of 0.242.

2.12 Experiment 12: Considering passive/active voice only for WHO-type Then we suspected it would be better if we considered only voice for WHO-type questions, as WHO-type questions contained more questions in which voice was crucial (for example, in the cases like “Who killed Lincoln?” vs “Who was killed by Lincoln?”) This experiment performed better than the previous one, but still was outperformed by Experiment 6. It resulted in 0.252. We think the reason for this is the same as explained in Experiment 10.

2.13 Experiment 13: SRL 10% Then we explored Semantic Role Labeling. Using SRL tools in practnlptools(https://pypi.python.org/pypi/practnlptools/1.0), we experimented whether using SRL helped with performance. For this experiment, we gathered all the semantic roles the question words were tagged with (Who, when, and where). For example, if Who was tagged ‘R-A1’ for verb use, we recorded A1. If there were multiple verbs in sentences, all the verbs and the corresponding semantic roles were recorded. Then, after retrieving sentences, we tried to improve the ranking procedure by providing a bonus to candidate sentences that had the same verb with the corresponding semantic roles found in the question. If there was a match in the candidate sentence, it was given a bonus of 10 percent. Unfortunately, this also underperformed compared to Experiment 6, which only had keyword ranking. Its MRR score was 0.287.

2.14 Experiment 14: SRL 30% We tried a similar experiment with 30% as the bonus score, but it performed even worse than the previous experiment, with 0.285. We concluded that our simple uses of syntax/semantic role labeling did not improve on the keyword ranking approach. We chose not to include these features in further experiments.

2.15 Experiment 15: sorting answers After examining thoroughly on the output answer file from Experiment 6, we then realized that we are being naive on the ordering of answers that all the answers will be put in the same order as they appear in the sentence. We definitely would prefer a more sensible ranking on answers. A strategy is to sort the answers by its distance to the keywords. The closer it is to keywords, the more relevant it is supposed to be. We decided to formularize the distance to keywords as the average of its distance to each keyword in the sentence, since often there are more than one keyword in the candidate sentence. Upon implementation, MRR slightly to 0.296. We chose to keep this implementation for further experimentation.

2.16 Experiment 16: fine tuning with window size and number of retrieved passages Then, we experimented with sentence retrieval window sizes. So, instead retrieving only the sentence with matching NER tag and keyword, we also included the sentence that followed it. This increased performance to 0.298.

We then tried to include 4 following sentences instead of just 1 so that more information will be retained in . This, however, led to a decrease in MRR to 0.283. We think it might have included too many irrelevant sentences.

In addition to having 2 as our sentnece retrieval window size, We also tried to remove the upper limit for number of retrieved passages, instead of 30 we had previously had for performance reasons. and the MRR went a tiny bit up to 0.299. Considering computational efficiency, we ultimately think the system limiting number of answers to 30 (0.298) will be overall the best.

Note: The best MRR we obtained is from the experiment where the number of answers is not limited, so for the final `answer_dev.txt` that we submitted where only 5 answers are retained, the MRR is decreased to 0.285.

3 A walk-through of QA processing

To better illustrate our system, we give a walk-through of how our system handles a question and outputs a list of answers. Consider the first question, Q89 "Where is Belize located?". First of all, the question will be classified as a WHERE-type question, hence the system will look for answers with LOCATION, FACILITY, and GPE taggings. Next, the list of keywords will be obtained as "Belize" and "located", after the stop words and punctuations are eliminated. The system will then start scanning sentences from document 1 onward. All sentences that has as least one word with desired NER tagging and at least one word sharing the same stem as either "Belize" or "located" will be retained. The sentence following the matching sentence will also be retained. For example, the sentence in document 1 "Belize suspends territorial pact with Guatemala" and "Belize has also asked the Guatemalan government for a review of the pact" will be retained, so as their following sentences. We will stop scanning once we have 30 passages (consisting of the leading sentence and following sentence). Then each sentence will be evaluated for its similarity to the keywords, and we will then sort them by similarity from high to low. For each sentence in order, we will extract answers with matching tags. After obtaining answers from all sentences, we sort the answers by its distance to keywords. Note that if we obtained more than 5 answers, we only keep the top 5; also if we run into duplicate answers, e.g. "Guatemala"", we will only consider the distance to keywords for the first occurrence of"Guatemala". Consequently, all answers will then be added to the answer text file.

4 Result analysis

Based on the MRR score 0.298, our system beats the baseline system (MRR 0.084) for sure. Also, the final system has 103 questions with no answers found in top 5 responses, compared to the baseline system where 204 questions had no answers found in top 5 responses. This is due to several advantages of our final system over the baseline:

- The relevance of sentences is taken into consideration (this is why for Q110, the expected answer "Portugal" didn't turn up in baseline system's answers but did in the final system's answers)
- The time tagging has been taken care of
- The relevance of answers is also considered (this is why for Q119, the expected answer "Cheops" is ranked the first, whilst it is not found in the baseline system's answers)
- Considering word stems for keywords also improved the system's performance greatly
- Adding the next sentence is also a good factor, but this is not groundbreaking since the answer will be sorted by distance to the keywords after all, so if the right answer is in the next sentence, it still has a big chance to be ignored due to its long distance to the keywords.

Certainly the system has drawbacks. For example, Q96 "Who killed Martin Luther King?" has the right answer 'Ray' in document 7, but because we have retrieved enough sentences from the previous documents, we did not scan up to document 7 yet, thereby missing the right answer. Also, the way to sort answers is not always perfect. For example, Q99 "Who is the Greek God of the Sea?" has the right answer 'Poseidon' which can be found in document 5 "Six moons of Neptune discovered by the Voyager 2 spacecraft in 1989 have been named after water nymphs and the children and lovers of the mythological *Greek sea god Poseidon*" but the system only ranked 'Poseidon' 21st, whilst 'Howard' is ranked the 1st extracted from document 3 "Think of the East Aegean sea by night ..', Howard is magnificent". This is because the average distance to keyword(s) is tied here. And the system by default will rank the earlier answer higher.

Overall, our final system is outperforming the baseline system a lot. It successfully picked up answers to 101 more questions than the baseline system, and increased the accuracy (MRR) to 0.298. It can catch the right answer with a higher chance (such as Q110 and Q119) due to its consideration on relevance of the sentences and answers to the question. We learned that NER tagging was essential, as this improved performance by a lot. For sentence retrieval, we learned that ranking by keyword worked really well, and it outperformed any other schemes that we tried. We tried hard and experimented to improve the performance even more by venturing into dependency parsing and semantic role labeling, but we weren't successful. Perhaps using a more sophisticated scheme with dependency parser and SLR would have worked better, but we learned that using syntax/semantic role required much more sophisticated schemes than we tried.

5 Work distribution

For part 2 of this project, Saerom did coding for experiments 1 (partially),2,4,5,6,15; Gi Yoon did coding for experiments 7-14; Rui did the coding for experiment 1 (partially) and 3. Xiaoyun wrote up the report. We discussed about experimental designs all together through the entire course.