Report: QnA Tool

Saem Habeeb: 2022MT62004, Anup Nayak: 2022CS51827
Aditya Mishra: 2022MT11271, Vidushi: 2022CS11095

1. Introduction

This report elaborates the functionality of the QnA tool created for the Assignment 7 of the course COL106. Briefly, this tool takes in a question, processes it and ranks the paragraphs in a specific Corpus based on the question. Once ranked, it feeds the top paragraphs to a LLM (ChatGPT) which provides a suitable answer based on the question. This tool has been fine-tuned for the corpus - *The Collected Works of Mahatma Gandhi* which contains 98 books but with subtle changes can be tuned to work effeciently on any corpus.

2. Working

2.1. Processing the Corpus

We were provided with a .csv file containing the frequencies of words in a general setting. We used a Trie to store the words with their frequencies giving us efficient *O(length of word)* time for retrieving frequencies. Further we also created a .csv file using *Dict* from A6 for the whole *98 book corpus* which stores the word and their frequencies in the books. This .csv file is used to create a Trie for the corpus each time a QNA_tool object is constructed.

A vector of para (paragraph) objects is also created with the QNA_tool object. Each para object stores the following:

- Location: Book code, Page number, Paragraph number
- Score (used to rank paragraphs)
- A vector which stores the frequency of words in the paragraph along with the word. Index of a particular word is found via hashing.

2.2. Processing the question

Generally nouns are more important as keywords in a question. But the nouns can be found in plural as well as singular forms. We have created a function **Singularize(string& word)** which takes a word as an input and returns a vector of it's possible singular forms if it is plural (and ends in an 's') or just returns a vector with one word. Different cases of plurals such as "ves", "ies", "es" and "s" ending plurals have been taken into account.

We have analysed the unigram_freq.csv and observed that most of the words having $frequency \geq 90M$ can't be keywords for a question. The exceptions are stored in the *words_to_be_included.txt* file. A vector of these words (named as *words_not_ignorable*) is created using the .txt file when a QNA_tool object is created.

Once we have received the question in the QNA_tool::query function, the following steps are performed on it to extract the important words in the question: (The important words are stored as a vector of strings)

1. The question is broken into words based on the separators (.,-:!"'()?[];@). The words are stored as a vector of strings.
2. For the rest of the words, the following is done.
3. If the word or any of it's possible singular forms appears in the *words_not_ignorable* vector, append it to the vector of important words.
4. Else If the word is "Mahatma" or "Gandhi" or "Gandhiji" or "India" or some other common word is ignored.
5. Else If the word or any of it's possible singular forms have $frequency \geq 90M$ in a general setting it is ignored.
6. Else If the word contains digits then it is possibly a date/year so it is important. Append it to the vector of important words twice so that it is weighed more while calculating a paragraph's score.
7. Else If the first letter is capital, then append it to the vector of important words twice.
8. Else the word maybe important, so append it to the vector of important words.

After all these steps if the vector of important words is found empty, just ignore the words with $frequency \geq 90M$ and add the rest to the vector of important words.

Still if the vector is found empty, consider all the words in the question string as important.

Form a space separated string of the important words and rank the paragraphs based on this string.

2.3. Deciding k and Getting top k paragraphs

Fixing a particular k for all queries might lead to the top-k paragraphs containing too less information or excess of information. Surely k shouldn't be too large. Lets place the upper bound of 15 for k. In our model, we have dealt with it by doing the following:

1. Once the score for all the paragraphs is calculated, sort the vector of para (paragraph) objects using Heap-sort till top 15 paragraphs are obtained.
2. Create a linked list for returning the location of the top-k paragraphs.
3. Insert the paragraphs' locations in the linked list till (total score of inserted paragraphs) l 5 * (highest score) or 15 paragraphs are inserted.
4. Return the head pointer of the linked list along with k.

Considering *gpt-3.5-turbo* can only process 4096 tokens at once, we feed it the paragraphs till all the paragraphs are used or 4096 tokens are fed.

3. Conclusion

This model can be extended to automize research work which requires lots of manual work for big firms, can also be used by students to do revision work for course material and has lots of other use cases. This project helped us understand the importance of using efficient data structures and learning useful algorithms.