COL106, SUMMER 2023

Assignment-7 REPORT

Jakharia Rishit (2022CS11621)
Dhruv Chaurasiya (2022MT11172)
Karan Deo Burnwal (2022EE11342)
Aditya Jha (2022CS11102)

INTRODUCTION

The two parts of this assignment were –

a) Based on the input query, retrieve the relevant paragraphs and core them based on relevance.
b) Use the ChatGPT API to answer the query based on the scored paragraphs.

PROBLEM DESCRIPTION AS UNDERSTOOD

a) The input to the code is the corpus of 98 files containing the works of Mahatma Gandhi and a query.
b) We use the query to find relevant paragraphs by searching from the corpus and retrieving relevant paragraphs.
c) Now we feed these paragraphs and the input query as a prompt to chat-GPT along with instructions to answer the input query in the most accurate way

ALGORITHM

a) Data structure stores book_code, page, and paragraph in the vectors in this order, and SymbolTable is an AVL-Tree

```
vector<vector<vector<SymbolTable>>> mainobj;
```

b) Input is taken through the following lines of code

```
void QNA_tool::insert_sentence(int book_code, int page, int paragraph, int sentence_no, string sentence)
{
    if (book_code >= mainobj.size())
    {
        mainobj.resize(mainobj.size() + 10);
    }

    if (mainobj[book_code].size() <= page)
    {
        int c = (512 > page) ? 512 : page;
        mainobj[book_code].resize(c + 3);
    }

    if (mainobj[book_code][page].size() <= paragraph)
    {
        int c = (16 > paragraph) ? 16 : paragraph;
        mainobj[book_code][page].resize(c + 3);
    }

    vector<string> words = ppr(sentence);
    for (string k : words)
    {
        mainobj[book_code][page][paragraph].insert(k);
    }

    // Implement your function here
    return;
}
```

c) Scoring Algorithm is -> First, we score the sentences using the frequency of the words that appear in the query and the paragraphs, then we push the word along with
(occurrence of the word in the sentence)/(total occurrence of the word)

```cpp
vector<pair<string, double>> QNA_tool::scoring(string sentence)
{
    vector<string> words = ppr(sentence);
    vector<pair<string, unsigned long long>> result;
    vector<pair<string, double>> resoto;

    for (int i = 0; i < words.size(); i++)
    {
        unsigned long long totalcount = 0;
        for (size_t j = 0; j < mainobj.size(); j++)
        {
            for (size_t k = 0; k < mainobj[j].size(); k++)
            {
                for (size_t l = 0; l < mainobj[j][k].size(); l++)
                {
                    totalcount += mainobj[j][k][l].search(words[i]);
                }
            }
        }
        result.push_back(make_pair(words[i], totalcount));
    }

    for (int i = 0; i < result.size(); i++)
    {
        ifstream frequency_file("unigram_freq.csv");

        std::string word1;
        std::string countofword;

        getline(frequency_file, word1);

        while (getline(frequency_file, word1, ','))
        {
            getline(frequency_file, countofword);

            if (word1 == result[i].first)
            {

                double c = (double)(result[i].second) / (double)(stod(countofword));
                resoto.push_back(make_pair(result[i].first, c));

                break;
            }
        }

        frequency_file.close();
    }

    return resoto;
}
```

d) This scored sentence is then used to score the paragraphs and then we output the get_top_k paragraphs using this scoring

```cpp
vector<QNA_tool::scored_para> QNA_tool::score_paragraphs(string sentence)
{
    vector<scored_para> ret;

    vector<pair<string, double>> scores = scoring(sentence);

    for (size_t i = 0; i < mainobj.size(); i++)
    {
        for (size_t j = 0; j < mainobj[i].size(); j++)
        {
            for (size_t k = 0; k < mainobj[i][j].size(); k++)
            {
                double __score = 0;
                for (size_t w = 0; w < scores.size(); w++)
                {
                    __score += ((double)mainobj[i][j][k].search(scores[w].first)) * scores[w].second;
                }

                scored_para qq;
                qq.book_no = i;
                qq.page_no = j;
                qq.para_no = k;
                qq.score = __score;

                ret.push_back(qq);
            }
        }
    }

    return ret;
}
```

e) Now we use the query function, and we feed the question and filename into the function; now we extract the top k paragraphs using the get_top_k_para and feed this information to query_llm

```cpp
void QNA_tool::query(string question, string filename)
{

    string API_KEY = "sk-szIWz8k1y3jcVs3KY1o8T3B1bkFJXTQfTMJtrNOKjEccV3n2";

    int k_paragraphs = 4;

    vector<string> q = ppr(question);

    string ques = "";

    for (auto i : q) {
        ques += i + " ";
    }

    Node* root = get_top_k_para(ques, k_paragraphs);

    question += "\nAnswer in about 300 to 400 words using only the excerpt as your source of information,
                given to you in DECREASING ORDER OF PRIORITY, treat them accordingly. YOU HAVE TO ANSWER IN A SINGLE PARAGRAPH.\n";

    query_llm(filename, root, k_paragraphs, API_KEY, question);

    return;
}
```

PROMPT ENGINEERING

a) As a group, we have decided not to disclose our prompt engineering techniques. It is an exceptional and rare technique developed with hard work and **consistent** efforts.
P.S. Just kidding, it's up there.
We have forced the GPT to only use the given excerpts to give answers of a specific word count range and in a single paragraph.

THANK YOU!