Python project

Summary:

I. Introduction

In the dynamic landscape of artificial intelligence and natural language processing, the development of chatbots and generative artificial intelligences has become a focal point for creating interactive and intelligent systems. This project embarks on a journey into text analysis, steering away from the intricacies of neural network manipulation. Instead, it focuses on a pragmatic approach rooted in the frequency of word occurrences to generate intelligent responses from a given corpus of texts. This methodology provides a fundamental understanding of the core concepts employed in text processing, laying the groundwork for the creation of effective question-answering systems.

II. Methodology

A) Data pre-processing

The data pre-processing process constitutes the crucial first step in this text analysis project. Its primary objective is to enable the program to collect and prepare a set of documents to better understand the nature of their content. This phase is of fundamental importance as it directly impacts the quality of subsequent analyses and the relevance of responses generated by the system.

B) Document collection

The first sub-step involves collecting a set of documents that will form the corpus on which the system will operate. These documents can originate from various sources, ranging from academic texts to conversational corpora, depending on the specific project requirements. Special attention is given to the diversity and representativeness of the selected documents to ensure adequate coverage of potential topics.

C) Text cleaning

Once the documents are collected, the program undertakes a text cleaning phase. This operation aims to eliminate undesirable elements that could introduce noise into the analysis. To achieve this, punctuation is removed, letters are converted to lowercase for normalization, and the text is segmented into individual words or "tokens." This process ensures uniformity in text processing, thus facilitating subsequent analysis steps.

D) Response preparation

The pre-processing phase extends beyond understanding the documents; it also lays the groundwork for generating subsequent responses. By comprehending the content of the documents, the program can identify recurrent patterns, key terms, and word associations that will be crucial for the response mechanism. This preparation optimizes the selection of potential responses in the subsequent steps.

E) Importance of pre-processing

Data pre-processing plays a crucial role in the final quality of the system. By cleaning the text and understanding the nature of the documents, the program ensures that subsequent analyses are based on reliable and representative data. This also enables the system to better adapt to the diversity of questions it might encounter, strengthening its ability to extract relevant information from the corpus.

In conclusion, this method establishes the necessary foundations for a successful text analysis. It allows the program to comprehend the context, clean the text for precise analysis, and prepare responses based on the nature of the corpus. These initial steps are critical for the overall success of the text analysis and intelligent response generation project.

III. Description of our project did

To build the project, we have followed this methodology. Step by step, we created each function one by one. In organizing the program, we have delineated it into two major components: the main and the functions. Separating program into 2 parts, it becomes apparent that each function aligns more or less with a step outlined in the methodology. For examples, we can see a function that create the final TF_IDF matrix of the entire corpus (1) and another that find the most important word of the question in the president speeches (2). These are the two most important functions of the program.

(1)

```
def tf_idf(dico_tf, dico_idf):
    """
    Calculate the TF-IDF matrix for the corpus.

    Parameters:
    dico_tf (list): List of dictionaries, each containing the term frequency for words in a file.
    dico_idf (dict): Dictionary with words as keys and their IDF scores as values.

    Returns:
    list: TF-IDF matrix, where each row represents a word, and each column represents a file.
    """
    matrix = []

    # Iterate over words in the IDF dictionary
    for word in dico_idf:
        row = [word]

        # Calculate TF-IDF score for each file
        for i in range(len(dico_tf)):
            row.append(dico_tf[i][word] * dico_idf[word])

        # Append the row to the matrix
        matrix.append(row)

    return matrix
```

(2)

```
def find_the_sentence_in_speeches(relevant_file, relevant_word):
    """
    Find the sentence containing the relevant word in a specific document.

    Parameters:
    relevant_file (str): Name of the relevant document file.
    relevant_word (str): Relevant word to search for in the document.

    Returns:
    str: The sentence containing the relevant word in the specified document.
    """
    with open("./speeches/" + relevant_file, "r", encoding="utf-8") as f:
        content = f.read()
        list_of_words = content.split()
        index = 0
        found = False
        word_uncleaned = ''

        # Iterate over words in the document content
        while not (found) and index < len(list_of_words):
            cleaned_word = clean_text(list_of_words[index])

            # Handle cases with multiple words separated by spaces
            if " " in cleaned_word:
                mini_list = cleaned_word.split()
                for elt in mini_list:
                    if elt == relevant_word:
                        found = True
```

```
                    word_uncleaned = elt
            elif cleaned_word == []:
                pass
            elif cleaned_word[0] == relevant_word:
                found = True
                word_uncleaned = cleaned_word[0]
            index += 1

    # Split content into sentences and find the one containing the relevant word
    text_splited_in_sentence = content.replace( old: "M.",  new: "Mr").split(".")
    for sentence in text_splited_in_sentence:
        if word_uncleaned in sentence:
            return sentence + '.'
```

In most cases, the function as well as all variables have a very explicit name, which allows anyone to easily understand the set of data in the program. Furthermore, we have created, for each function of the branch, a standardized docstring to quickly recall or assist a new user wanting to test our script.

In the case of the "main" branch, we proceeded differently. Indeed, in this part, we mainly used the functions seen earlier to make the chatbot work. For example, we have:

```
question = tf_idf_question(tf_question(present_terms_in_corpus(corpus,clean_text(question_user))),idf_dico)
```

Also, we have decided to run the main branch with a while loop so that the user can stay with the chatbot for as long as they want. This while loop covers all the capabilities of the AI and then expresses the actions requested by the user. The choice of actions is entirely free for the user. Thus, our entire branch can be simplified into a single loop.

```
> while True: ...
```

We also made the choice to create an interface that is easy to access and understandable for everyone.

```
Hi, I'm Raquita ! My wonderful editors create me to analyse texts while they are studying hardly to success their DE in algebra (spoil: they failed).
    If you want to analyse texts enter: analyse,
    If you want to ask me some questions about texts enter: chat
    If you want to leave me enter: stop
What do you want to do ?
```

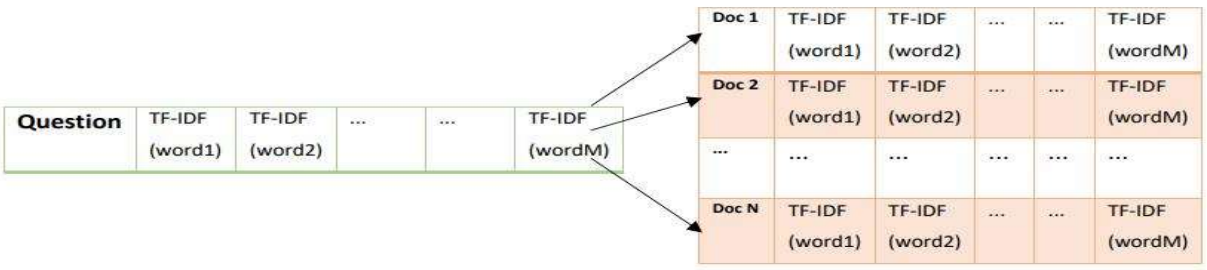
```
What do you want to do ? analyse
Choose the speeche you wish to analyze
                    1 - Nomination Chirac 1
                    2 - Nomination Chirac 2
                    3 - Nomination Giscard dEstaing
                    4 - Nomination Hollande
                    5 - Nomination Macron
                    6 - Nomination Mitterand 1
                    7 - Nomination Mitterand 2
                    8 - Nomination Sarkozy
Enter the number here : 3
Choose the function you want to use on the speech
                1 - president_name_exact
                2 - first_name_associate_name
                3 - cleaned_file
                4 - total_word
                5 - term_frequency
                6 - tf_idf
Enter the number here :4
messieurs
```

To conclude this section, we can deduce that we have truly chosen to build a program around the simplest possible use while retaining highly technical features.

IV. Problems encountered

The main problem we faced is GitHub. Indeed, it is impossible for us to commit our algorithms in our common repository. This has caused us to lose valuable time for the rest of the project. This issue is not specific to the project's code, but we felt it was important to highlight.

From a purely code perspective, the most problematic error was in creating the question vector. Indeed, each time we ran the program, the vector had either an unreadable size or a value that was far too small. So, for a few hours, we searched for the error in our script. After this long search, we realized that the matrix was not being read in the correct direction. It was by carefully rereading the subject's PDF that we understood our mistake with the diagram presented in the subject.



Overall, this is the only major code issue we encountered during our project. There were also some calculation errors, especially for IDF and the TF-IDF matrix, which were due to a misunderstanding of the instructions or their lack of precision.

V. Solutions

Unfortunately, we were unable to address the GitHub problem. However, we found a solution to our issue related to the misreading of the matrix. For this, we inserted a new function, "transpose matrix", to reverse the matrix so that it has as many columns as there are words in the corpus.

```
def transpose_matrix(matrix):
    """
    Transpose the input matrix.

    Parameters:
    matrix (list of lists): Input matrix.

    Returns:
    list of lists: Transposed matrix.
    """
    nb_row = len(matrix)
    nb_col = len(matrix[0])

    # Use list comprehension to create the transposed matrix
    transposed_matrix = [[matrix[i][j] for i in range(nb_row)] for j in range(nb_col)]

    return transposed_matrix
```

This function really helped us out. It was very useful for the question vector, and we also used it for subsequent questions to make the script faster, allowing it to avoid rereading all the documents.

VI. Conclusion

In conclusion, this project has been a rewarding experience that allowed us to deepen our knowledge in text analysis and the development of question-answering systems. Despite some challenges, such as GitHub issues and calculation errors, we successfully overcame these obstacles through effective collaboration within the team. Utilizing methods based on word frequency for generating intelligent responses added a practical dimension to our understanding of fundamental text processing concepts. By integrating technical features while ensuring a user-friendly interface, our aim was to create an accessible and efficient program. This project has strengthened our ability to tackle complex problems and apply theoretical knowledge in a practical context. Ultimately, this experience has significantly contributed to our learning and professional development.