Python project

My first chatBot

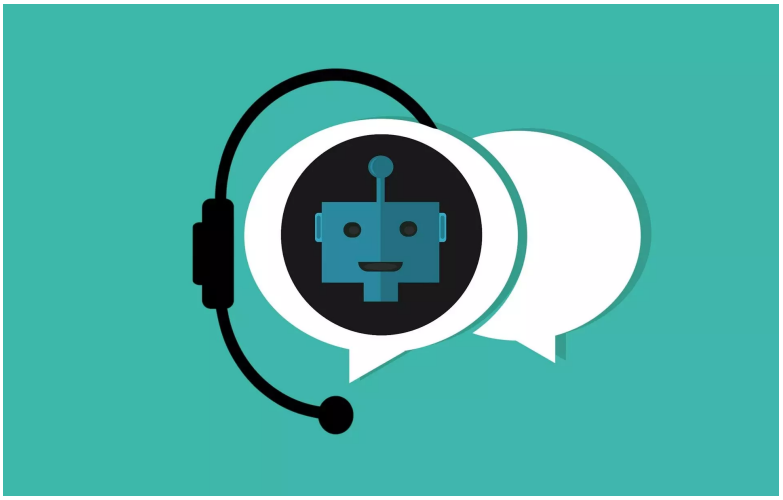
Written Assignment

INT4 - Group 2

Timothée BAUDRY

Oscar MASDUPUY

2023-2024



PARIS PANTHÉON-ASSAS UNIVERSITÉ

TABLE OF CONTENT

Welcome to My First Chatbot Project, a journey into the fundamental concepts of text processing and a deep dive into one of the methodologies employed in the development of chatbots and generative artificial intelligences like ChatGPT.

In this project, we will explore a method centered around word occurrences to generate intelligent responses from a given corpus of texts.

While we won't delve into the complexity of manipulating neural networks, our focus will be on a robust algorithm that harnesses the power of words and their frequencies to craft meaningful answers. The primary objective is to design a system capable of responding to questions based on the prevalence of words within a given corpus.

The project was focused on 4 main objectives:

-Data pre-processing

-Creating a TF-IDF matrix

-Similarity calculation

-Selecting the best answer

Note : All the codes presented in this report are commented for further understanding in the submitted code

The functioning of this chatBot is different from the functioning of the well known one like ChatGPT or BingAI. The initial stage involves gathering and preparing a collection of given documents(in our case presidents' speeches) for analysis to comprehend their content. This process includes text refinement through the elimination of punctuation, conversion of letters to lowercase, and segmentation of the text into individual words or "tokens."

This part is called the Data Pre-Processing

```python
def modify_text_file_to_lowercase(file_path: str, new_folder_path: str, new_file_name: str) -> None:
    with open(file_path, 'r',encoding='utf8') as file:
        text = file.read().lower()
    new_file_path = os.path.join(new_folder_path, new_file_name)
    with open(new_file_path, 'w') as file:
        file.write(text)
text_to_modify = name.list_of_files(name.folder,"txt")
for j in text_to_modify:
    modify_text_file_to_lowercase(f"./speeches/{j}","./cleaned",j)
```

This function serves to transform all capital letter into lowercase

```python
lowercase_letter = "abcdefghijklmnopqrstuvwxyzüéàâäåçêëèïîìôöòûùÿáíóúñ1234567890"
def ponct_changes(file_path: str):
    f = open(file_path, "r", encoding = 'utf-8')
    res = ""
    for i in f.read():
        if i in lowercase_letter:
            res+=i
        else:
            res += " "
    with open(file_path, 'w') as file:
        file.write(res)
```

This function serves to delete all capital type of punctuation

The next step in our program is the creation of what is called a TF-IDF Matrix. This works as so, first we create 2 functions, TF and IDF that calculate, respectively, how often a word appears and the importance of a word, in a document. After that, we can calculate the TF-IDF score of the word by multiplying both values. Every word in the corpus is linked to a vector, with its dimension corresponding to the total number of documents in the collection. This results in the formation of a TF-IDF matrix, where each row represents a word, and each column represents a document in the corpus.

```
def tf(string): #measures how often a word appears in a specific document
    dict = {}
    words = ""
    for i in string:
        if i == " " or i in ",;:!?./'-" or i == "\n" and words != "":
            if words in dict:
                dict[words] += 1
                words = ""
            elif words not in dict and words != "":
                dict[words] = 1
                words = ""
        if i not in ",;:!?./-' \n":
            words += i

    if words in dict:
        dict[words] += 1
    elif words not in dict and words != "":
        dict[words] = 1
    return dict
```

This function serves to measure how often a word appears in the speeches.

```
def idf(directory):  #measures the importance of a word in the entire corpus of documents
    files_names, types, dic = [], ".txt", {}
    word = ""
    directory = r"speeches" + "\\" + directory
    for namefile in os.listdir(directory):
        if namefile.endswith(types):
            with open(directory + "\\" + namefile, 'r', encoding = 'utf-8') as file:
                    for word in tf(file.read()):
                        if word in dic:
                            dic[word] += 1
                        if word not in dic:
                            dic[word] = 1
    for word in dic:
        dic[word] = math.log(8/dic[word])
    return dic
```

This function serves to measure the importance of a word in the whole set of speeches

```
def tf_idf(directory): #score of a word in a given document is a numerical vector that reflects both the frequency of the word
    matrix_td_idf = []
    lign = []
    dict_idf = idf(directory)
    for filename in os.listdir(r"speeches" + "\\" + directory):
        if filename.endswith(".txt"):
            with open(r"speeches + "","\\" + directory + "\\" + filename, 'r', encoding = 'utf-8') as file:
                dict_tf = tf(file.read())
                for word in dict_idf:
                    if word in dict_tf:
                        lign.append(dict_idf[word] * dict_tf[word])
                    if word not in dict_tf:
                        lign.append(0)
            matrix_td_idf.append(lign)
            lign = []
    return matrix_td_idf
```

This function uses the two precedent function to calculate the TF-IDF matrix

After this step, we can go onto the next one : Similarity calculation. This consists of computing the similarity between the vector representation of the question and the vectors associated with words in the corpus, employing methods such as cosine similarity or alternative similarity measures. This process facilitates the identification of words in the corpus that exhibit the highest similarity to the given question.

```python
def scalar_product(vectorA,vectorB): #returns the scalar product of the two vectors
    scalar = 0
    for i in range(len(vectorA)):
        scalar += vectorA[i] * vectorB[i]
    return scalar
```

```python
def norm(vector): #returns the norm of a vector
    norm = 0
    for i in range(len(vector)):
        norm += vector[i]**2
    return norm**(1/2)
```

```python
def cosine_similarity(vectorA, vectorB):
    cosine = scalar_product(vectorA, vectorB) / (norm(vectorA) * norm(vectorB))
    return cosine
```

Those functions serves to calculate the similarity of the question and word in the corpus of speeches

After this, the chatBot now has all the key to select the best answer corresponding to the question asked by the user, to do so it goes as follows : The chatbot detects words in the corpus that share high TF-IDF similarity scores with the question. Subsequently, it chooses the answer that incorporates the highest count of these identified similar words.

```python
def find_most_relevant(tfidf_matrix, tfidf_vector, file_names): #find the most relevant document in the cleaned directory
    similarities = cosine_similarity(tfidf_matrix, tfidf_vector)
    most_relevant_index = similarities.argmax()    "argmax": Unknown word.
    return file_names[most_relevant_index]

def get_speeches_file_name(cleaned_file_name): #get the equivalent document from cleaned into speeches
    return cleaned_file_name.replace("./cleaned", "./speeches")

def find_keyword_and_sentence(tfidf_vector, tfidf_feature_names, document): #Locate the word with the highest TF-IDF score
    keyword_index = tfidf_vector.argmax()    "argmax": Unknown word.
    keyword = tfidf_feature_names[keyword_index]

    sentences = document.split('". "')
    for sentence in sentences:
        if keyword in sentence:
            return keyword, '"' + sentence + '"'

    return None, None

def generate_response(question, answer, question_starters): #added a capital letter at the beginning and a final point and
    answer = answer[0].upper() + answer[1:] + "."

    for starter, model_response in question_starters.items():
        if question.startswith(starter):
            return model_response + answer

    return answer
```

This set of functions serves to allow the chatbot to know where to look for its answer and generate it

Lastly the chatBot will provide the answer to the user.

The main function that manages the whole program allow us to -Display the names of the studied presidents, display the least important words in all speeches, display the most used word in a specific speech, display the presidents that have said a specific word, display the first speeches to talk about a specific topic

For us, this project served as a practical application of our previously acquired knowledge, translating theoretical concepts from our coursework into actual code. The most demanding aspects revolved around integrating various functions, given that both of us employed different names and sometimes anticipated different outputs. Initially, this led to a time loss that we managed to overcome, ultimately completing the project within the designated time frame. Collaborating as a team required us to adapt from individual coding practices. The number of functions and the overall code size emphasized the need for better organization, as our initial code lacked proper structure. Beyond technical challenges, this project served as an initiation into the practical aspects of managing a coding project.