Assignment Report: PDF Analysis and Sentiment Visualization App

Suhail Ahmed

Objective:

The goal of this assignment is to design and implement a Streamlit application for analysing PDFs containing survey responses, extracting key information, translating content to English, and visualizing sentiment and formality scores.

Key Requirements:

1. Extract first and last names, age, continent, and opinions on Paris.
2. Translate responses to English.
3. Provide sentiment and formality scores.
4. Build a visually captivating Streamlit application for end-to-end processing.

Executive Overview:

1. PDF Upload and Display (**pages/app.py**):

- Users can upload a one-page PDF using Streamlit's file uploader.
- The selected page is converted to an image for visualization.
- Page content is displayed in a text area for analysis.
- The app allows users to analyse the content and switches to the next page for further processing.

2. PDF to Text Conversion (**pages/read.py**):

- Extracts relevant information such as date, time, gender, and responses from the PDF.
- Filters responses collected after March 2023.
- Displays the extracted information in a tabular format for user verification.
- Transitions to the next step, allowing users to translate and analyse sentiment.

3. Backend (**modules/models.py**):

- Translates non-English responses using Google Translator from the deep_translator library.

- Calculate sentiment scores for each sentence using a pre-trained BERT-base multilingual model from Hugging Face.

- Calculate formality scores for each sentence using a pre-trained Roberta-base model from Hugging Face.

- Sends results to the frontend (pages/analyze.py).

4. Translate and Sentiment Analysis (**pages/analyze.py**):

- Utilizes language detection to identify non-English responses.

- Fetches translations of non-English responses to English.

- Fetches sentiment and formality scores for each response.

- Marks responses collected before March 2023 as 'N/A'.

- Sorts results based on sentiment score.

- Displays the final results in a table, including date, time, gender, original sentence, sentiment, and formality scores.

Code Quality and Best Practices:

- The code is well-structured, organized, and follows best practices for readability.

- Utilizes external libraries for PDF parsing (PyPDF2), text translation (Google Translator), and sentiment analysis.

- Implements error handling for potential issues in PDF processing and translation.

- Uses Streamlit for building an interactive and user-friendly interface.

System Architecture Overview:

The architecture of the system can be classified as a client-server architecture, specifically a thin-client architecture with some aspects of microservices architecture.

1. Client Side (Thin Client):

- Users interact with the Streamlit application through a web browser.
- The browser renders the user interface provided by Streamlit and sends user input (e.g., file uploads, button clicks) to the server.

2. Server Side:

- The server hosts the Streamlit application, which includes the logic for processing PDFs, performing text extraction, translation, sentiment analysis, and data visualization.
- External services, such as Google Translator module for language translation, are accessed from the server.
- Each step in the PDF analysis process is encapsulated within modular components or functions, providing flexibility and maintainability.

3. Communication:

- Communication between the client and server occurs over the network, typically using HTTP.
- The server processes client requests, performs necessary computations, and sends back responses, which are then rendered by the client.

4. Data Flow:

- PDF files uploaded by users are processed sequentially through different stages: PDF to text conversion, translation, sentiment analysis.
- Intermediate and final results are stored in memory (e.g., session state variables) and displayed to users through the Streamlit interface.

Rationale for the Architecture:

1. Scalability: Thin-client architecture allows for easy scaling of the application by adding more client instances without the need to modify the server.

2. Modularity: Microservices-like decomposition facilitates maintainability, reusability, and scalability of individual components.

3. User Experience: Streamlit provides a user-friendly interface for interacting with the application, abstracting away complexities of the backend processing.

4. Flexibility: Separation of concerns between client and server allows for independent development, deployment, and scaling of frontend and backend components.

[The following pages discuss the technical overview]

<u>**Technical Overview:**</u>

PyPDF2 is used for reading PDF files, pdf2image for converting PDF pages to images. This image will be an image of the PDF document uploaded by the user; this will indicate to users that their document has been uploaded successfully.

The sentiment_formality_analysis() function is applied to each sentence in the DataFrame, resulting in a sentiment and formality score for each response. A BERT-base-multilingual-uncased model is used for sentiment and formality analysis. The model was finetuned for sentiment analysis on product reviews in six languages:

1. English
2. Dutch
3. German
4. French
5. Spanish
6. Italian

It predicts the sentiment of the review as a number of stars (between 1 and 5). This output is normalized to fit in the range of 0 to 1 to match the requirements.

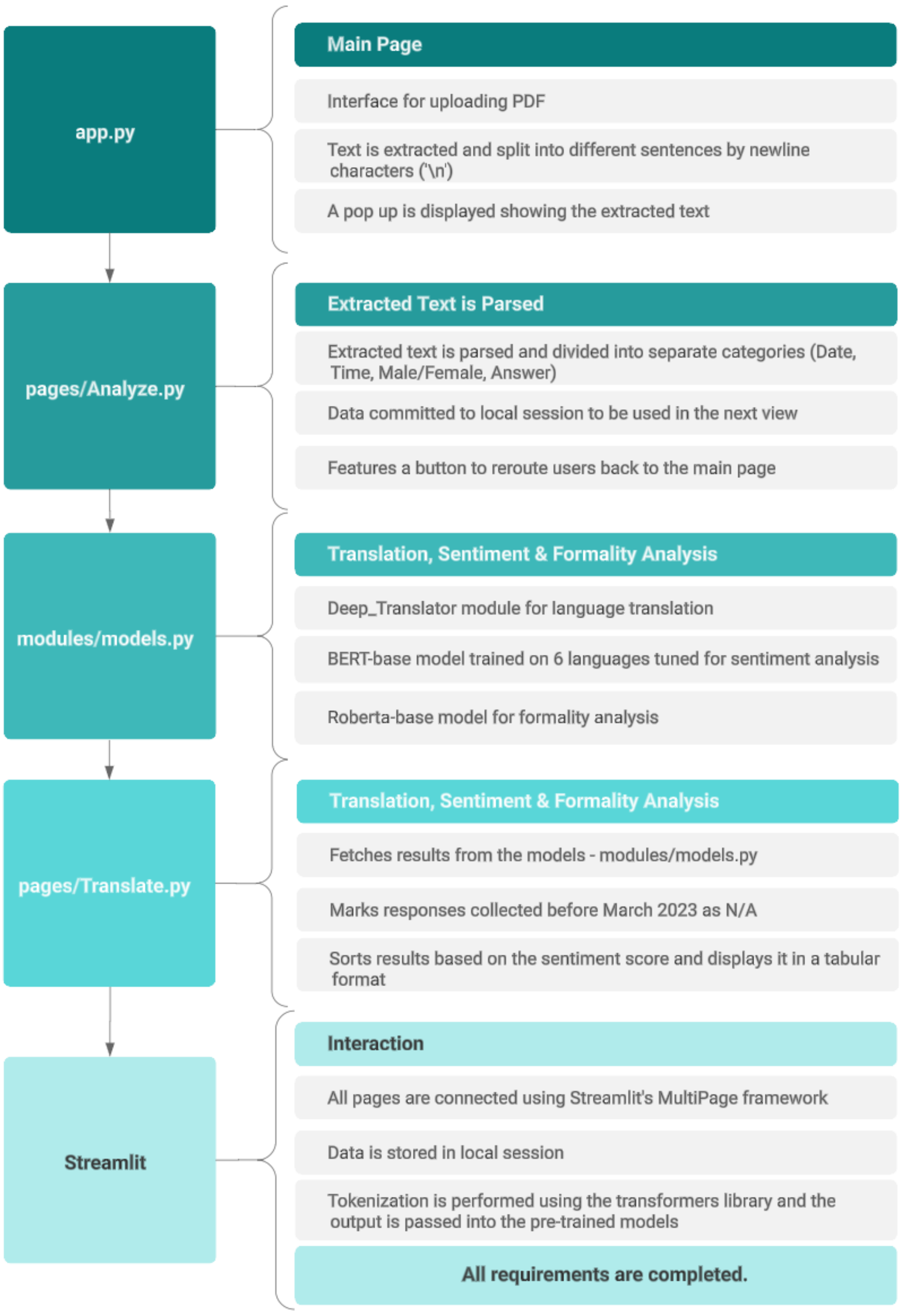
Sentences belonging to any of the 6 aforementioned languages are not translated to English before being pre-processed and analysed. Other languages are translated to English using the GoogleTranslator from the deep_translator module. The source language is auto detected when the text is passed and the target language is set to English.

The scores of responses collected before the 1st of March 2023 are marked as 'N/A'. The rest are displayed in tabular format, ordered by the sentiment score.

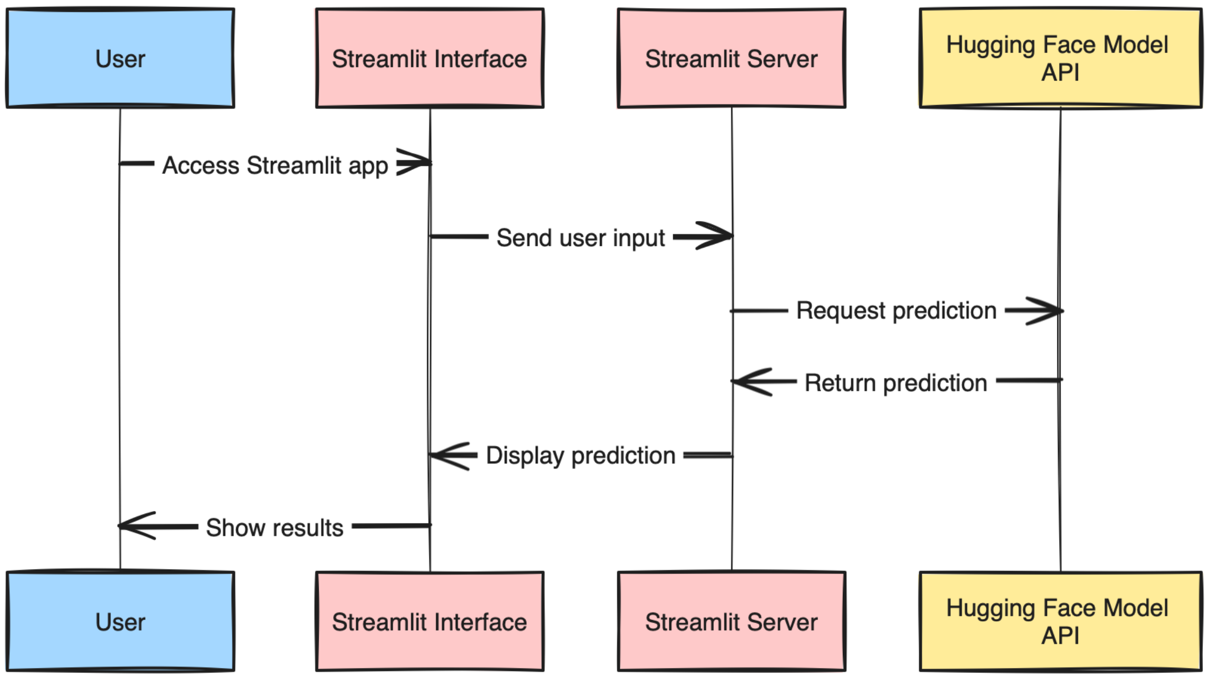
Pre-processing involves tokenization that is done using the AutoTokenizer function from the transformers library. This function tokenizes the string of text in accordance with the BERT-base model.

A Roberta-base model which was trained to predict for English sentences, whether they are formal or informal is applied. Each sentence is augmented by changing the text to upper or lower case; removing all punctuation, adding dot at the end of a sentence. It was applied because otherwise the model is over-reliant on punctuation and capitalization and does not pay enough attention to other features. These augmentations were done by using the RobertaTokenizer function from the transformers library.

Intra-File Communication:


app.py
Main Page
Interface for uploading PDF
Text is extracted and split into different sentences by newline characters ('\n')
A pop up is displayed showing the extracted text
pages/Analyze.py
Extracted Text is Parsed
Extracted text is parsed and divided into separate categories (Date, Time, Male/Female, Answer)
Data committed to local session to be used in the next view
Features a button to reroute users back to the main page
modules/models.py
Translation, Sentiment & Formality Analysis
Deep_Translator module for language translation
BERT-base model trained on 6 languages tuned for sentiment analysis
Roberta-base model for formality analysis
pages/Translate.py
Translation, Sentiment & Formality Analysis
Fetches results from the models - modules/models.py
Marks responses collected before March 2023 as N/A
Sorts results based on the sentiment score and displays it in a tabular format
Streamlit
Interaction
All pages are connected using Streamlit's MultiPage framework
Data is stored in local session
Tokenization is performed using the transformers library and the output is passed into the pre-trained models
All requirements are completed.

System Interaction Sequence Diagram:



Closing Remarks

1. Is the use case possible with current generation technologies?

Yes, the use case is feasible with current generation technologies. Many of the tasks described, such as text extraction, named entity recognition, sentiment analysis, and language translation, are supported by existing NLP tools and models. The specific implementation details and choice of models may vary based on the languages involved and the complexity of the data, but the general capabilities are available. For instance, a lot of sentiment is lost in translation when converted from to English from another language. Models trained on those languages would be preferred. The same struggle is faced for formality score computation as well.

2. Can sufficient human guard-rails be implemented to ensure quality of service and reinforcement learning while still delivering a meaningful improvement in productivity?

Yes, implementing human guard-rails is crucial for maintaining quality of service and addressing potential issues, such as biases or errors in automated processes. Combining automated systems with human oversight, especially in tasks like named entity recognition and sentiment analysis, can enhance the overall accuracy and reliability. Reinforcement learning can benefit from continuous feedback loops with human reviewers. Striking a balance between automation and human intervention is essential to ensure ethical deployment and improvements in productivity.

3. Is the problem real and general, or just an idiosyncratic issue with one organization?

The problem is real and general. Extracting information from diverse survey responses in multiple languages, performing named entity recognition, sentiment analysis, and translation are common challenges in NLP. Many organizations face similar tasks when dealing with multilingual and varied data sources. The need to ensure data quality, ethical considerations, and the interpretability of automated processes is a general concern in the deployment of NLP solutions and is not limited to a specific organization.