"Technical Approaches and Motivations Behind Backend and Frontend Development: A Comprehensive Report"

Streamlit Front-End

1. Overview

The front-end serves as the user interface for the data visualization web application. It allows users to upload CSV files, check file upload status, view basic dataset information (e.g., number of rows/columns, header), and potentially visualize the data's distribution. The front-end communicates with the backend FastAPI application for data processing and machine learning predictions.

---

2. Global Variables

- backend_url: The base URL of the FastAPI backend. This is used for making HTTP requests to interact with the backend (file upload, data processing, etc.).

---

3. Dependencies

The front-end relies on the following libraries:

- Streamlit: For creating the web application interface.
- Pandas: For data manipulation, particularly displaying dataset headers and information.
- Requests: For making HTTP requests to the backend.
- Tempfile: For managing temporary files used during the file upload process.

---

4. Components and Functionality

File Upload

- Function: upload_file()
    - This component handles the uploading of CSV files.
    - When the user uploads a CSV file, the app writes the file to a temporary path using tempfile.NamedTemporaryFile(delete=False).
    - The file is then sent to the backend via an HTTP POST request (/uploadfile/ endpoint).
    - After uploading, a request is made to store the temporary file path in the backend.

- Once the file is uploaded, a success message is shown with basic dataset information, such as the number of rows and columns.

Check File Status

- Function: check_file()
  - This component checks if a CSV file has been uploaded and processed in the backend.
  - It sends a GET request to the backend (/check_file/ endpoint).
  - Based on the response, it displays either an error message ("No csv file was uploaded") or a success message with the uploaded file's name.

Data Display and Information

- After uploading the file, the app displays the dataset header as a DataFrame using st.dataframe(). This gives the user a quick overview of the uploaded data's columns.
- The number of rows and columns in the dataset is also displayed using the information received from the backend.

Placeholder for Data Visualization

- Function: visualize_distribution()
  - This function is currently a placeholder. It is intended to allow users to visualize data distributions (e.g., histograms, box plots) after uploading a dataset.
  - Further implementation can be done here to interact with the dataset and generate various plots based on user selection.

---

5. Running the Application

To run the Streamlit application:

1. Ensure all dependencies (Streamlit, pandas, requests) are installed. This can be done via pip: pip install streamlit pandas requests
2. Save the Streamlit script in a Python file (e.g., app.py).
3. Execute the following command to run the application:

streamlit run app.py –server.address localhost –server.port 8501

4. Once the app is running, access it in the browser at: http://localhost:8501.

---

6. Styling and Customization

The app includes custom CSS styles that animate the title when the page loads. This style is defined in the h1_style variable. The @keyframes animation creates a sliding effect for the heading, which is applied to the <h1> tag on the page.

This adds an interactive element to the header, enhancing user experience.

FastAPI Back-end

1.Overview

The backend serves as the foundation for a data visualization web application. It facilitates tasks such as uploading CSV files, preprocessing data, making predictions using machine learning models, and interacting with a conversational AI pipeline.

2.Global Variables

- global_df: Holds the uploaded dataset as a Pandas DataFrame.
- global_filename: Stores the name of the uploaded dataset file.
- global_temp_path: Path to the template file used for processing.
- X_train, X_test, y_train, y_test: Variables to store preprocessed training and testing data.
- llm_pipeline: Holds the language model pipeline instance.

3.Dependencies

The application relies on the following libraries:

- FastAPI: For creating API endpoints.
- Pandas: For data manipulation.
- NumPy: For numerical computations.
- LangChain: For conversational AI and vector embeddings.
- Transformers: For loading language models and pipelines.

- Scikit-learn, XGBoost, CatBoost: For machine learning models.
- Pickle: For loading pre-trained models.

4.Endpoints

Root Endpoint

GET /

- Description: Returns a welcome message.
- Response: { "Message": "Data Visualiser Web App" }

File Upload

POST /uploadfile/

- Description: Uploads a CSV file to the server.
- Request: A file input (CSV format).
- Response:
    - number_of_rows: Total rows in the dataset.
    - number_of_columns: Total columns in the dataset.
    - header: First few rows of the dataset as a dictionary.

Check File

GET /check_file/

- Description: Verifies if a CSV file has been uploaded.
- Response:
    - Message: Filename if a file exists, otherwise "No csv file was uploaded".

Template File Path

POST /tempfilepath/

- Description: Accepts the file path for the template CSV.
- Request: JSON with the key temp_file_path.
- Response: Confirms the stored file path.

GET /tempfilepath/getpath/

- Description: Retrieves the stored template file path.
- Response: temp_file_path if available, otherwise null.
-

LLM and Vector Embeddings

POST /get_chain_result/

- Description: Processes a conversational query using a retrieval-based chain.
- Request:
  - question: User question.
  - chat_history: A list of conversation tuples (user prompt, system response).
- Response: The generated result from the conversational chain.

Feature Extraction and Prediction

POST /extract_features/validate/make_a_prediction/

- Description: Extracts features from user input, validates them, and makes a prediction.
- Request:
  - user_input: Free text from which features will be extracted.
- Response: Prediction result.

Churn Data Preprocessing

POST /ChurnUseCase/preprocessing/

- Description: Preprocesses the uploaded dataset for churn analysis.
- Response: Preprocessed training and testing data as lists.

Classification Methods

POST /classification_method/

- Description: Runs a classification model on the preprocessed data.
- Request:
  - model: Model type (e.g., xgboost, catboost, randomforest).
  - params: Hyperparameters for the model.

- Response:
    - cm: Confusion matrix.
    - acc_score: Accuracy score.
    - k_cross_score: Cross-validation score.
    - report: Classification report.

5.Utility Functions

load_llm()

- Loads the language model pipeline.

llm_chain()

- Sets up a conversational chain with LLM and vector embeddings.

generate_response()

- Generates a response from the LLM.

make_prediction()

- Loads a pre-trained model and makes a prediction using extracted features.

validate_and_complete_features()

- Ensures all required features are present and fills missing values with defaults.

6.Running the Application

To start the FastAPI application:

1. Ensure all dependencies are installed.
2. Execute the following command:
3. uvicorn app:app --host localhost --port 8000
4. Access the API at http://localhost:8000.

Technical and Business Motivations

A-Technical Motivations:

1. Seamless Integration of Components:

-The backend system is built with FastAPI, providing a lightweight, asynchronous web server capable of handling complex data interactions efficiently.

- The frontend system leverages Streamlit for an interactive user interface, ensuring end-users ease of use.

2. AI Integration:

-The use of Hugging Face's LLMs (meta-llama/Llama-2-7b-chat-hf) enables sophisticated text generation and comprehension capabilities. This is vital for tasks such as feature extraction and predictive insights.

-The FAISS (Facebook AI Similarity Search) indexing ensures efficient retrieval of relevant information from vectorized data.

3. Data Processing and Predictive Modeling:

-Incorporates robust preprocessing modules (ChurnPreProcessing) to clean and split datasets effectively.

-The machine learning pipeline supports various classification algorithms (e.g., XGBoost, CatBoost, Random Forest) with hyperparameter tuning, allowing for customized predictions.

4. Scalability and Flexibility:

-Modular architecture ensures that components can be updated or replaced without disrupting the overall system.

-Dynamic handling of models and parameters facilitates testing and deployment across multiple use cases.

B-Business Motivations:

1. Customer Retention:

- The predictive churn analysis directly supports decision-making for customer retention strategies by identifying at-risk users.

2. User Empowerment:

- A user-friendly interface provides businesses with actionable insights without requiring in-depth technical expertise.

3. Data-Driven Decision Making:

- Leveraging LLMs for feature extraction transforms raw data into actionable metrics, enabling strategic decision-making.

C- Alignment with Client Requirements:

1. Requirement: Predictive Modeling for Churn Analysis

- Alignment: The solution integrates a robust machine-learning pipeline with advanced classification algorithms to predict churn with 80% accuracy.

2. Requirement: Interactive and Intuitive User Interface

- Alignment: The Streamlit-based front-end provides an accessible and visually engaging interface for interacting with the model.

3. Requirement: Real-Time Insights and Query Responses

- Alignment: The backend supports real-time data queries, conversational AI, and predictive modeling, offering instantaneous responses.

4. Requirement: Scalability and Modularity

- Alignment: The architecture separates data handling, model training, and user interaction, allowing easy integration of new datasets, models, or features.