Leaf Identification App: Software Engineer

ROSHANAK BEHROUZ
Università degli Studi di Trieste
Piazzale Europa, 1
Trieste, TS. Italy.34127
Roshanak.Behrouz@studenti.units.it
Roshanak Behrouz

ABSTRACT

Software engineer play a pivotal role in this project, developing a web application that allows users to input specific leaf parameters and receive immediate species class number and converting it to leaves' scientific name. The application is built on a foundation of Python, utilizing libraries such as Pandas for data manipulation, Scikit-learn for model implementation, and Joblib for model serialization, ensuring efficient data handling and model management. This project not only aids academic and research-focused endeavors but also engages the general public in citizen science by making botanical classification accessible to non-specialists.

Keywords

Leaf Identification, Machine Learning operations, Python, Web application.

1. PROBLEM DEFINITION

The project's objective is to develop a user-friendly web application that facilitates the identification of leaf species through user-provided morphological features.

2. BACKGROUND

The "Leaf Species Identification" App integrates various technologies to facilitate classification using a machine learning approach. Key technologies include the RandomForestClassifier for accurate classification of leaf species which is performed by data engineer, and Pandas for efficient data handling and preprocessing. Streamlit was used to develop a user-friendly web interface, allowing users to interactively input data and receive species classifications. Additionally, the project utilized Joblib for model serialization, ensuring the machine learning model could be easily saved and reloaded for future predictions without retraining.

3. METHODOLOGY

I developed a web application using Streamlit, a popular Python library that facilitates the creation of interactive web interfaces for data-driven projects. Streamlit was chosen as the framework to develop the web interface due to its simplicity and effectiveness in turning data scripts into shareable web apps [2]. It enabled the rapid creation of a user-friendly interface where users can input leaf features and receive predictions. Its ability to handle both the front-end and back-end aspects of web development without the need for separate files or complex web frameworks significantly streamlined the development process. Figure 1 shows how it runs.

```
C:\Users\RB\Desktop\MLOPs\Final Project>streamlit run mlops.py

  You can now view your Streamlit app in your browser.

  Local URL: http://localhost:8501
  Network URL: http://192.168.1.9:8501
```

Figure 1: How To Run Streamlit

This application is specifically designed for leaf species identification based on morphological features. The script begins by importing necessary libraries: streamlit for the web interface, joblib for loading the machine learning model, and numpy for numerical operations. The script loads a pre-trained machine learning model from a file named 'model.joblib' (figure 2). This model is used to predict the species of a leaf based on various input features. The web interface is set up with the title 'Leaf Identification'. Additionally, an image of a leaf is displayed in the sidebar to visually enhance the application and provide context to the users. The application defines a series of sliders for user input, corresponding to different leaf features such as Eccentricity, Aspect Ratio, and others. Each feature has a defined range, and the sliders are arranged in rows of three columns for a clean and organized interface.

```
# Load the trained model and dataset
model = load('model.joblib')
```

Figure 2: How To Save A Model Using Joblib

Users adjust these sliders to input the values of the leaf features they wish to analyze. Figure 3 shows a sample of slider. A 'Predict' button is provided. When pressed, the application collects all the feature values from the sliders, formats them into an array, and feeds them into the machine learning model for prediction.

```
with tab1:
    st.header("Identify Leaf Type using Sliders")
    features = {
        "Eccentricity": (0.0, 1.0),
        "Aspect Ratio": (1.0, 20.0),
```

Figure 3: How To Create Slider

The result, which is the predicted leaf class, is then displayed to the user. In another section of the application, users can manually input feature values through text input fields instead of sliders. This provides an alternative method for entering precise values.

```
with tab2:
    st.header("Identify Leaf Type using Manual Entry")
    inputs = []
    for feature in features:
        inputs.append(st.number_input(f'Enter {feature}', format="%.5f"))
    if st.button('Predict Leaf Type Manually'):
        inputss = np.array(inputs).reshape(1, -1)
        predictions = model.predict(inputss)
        st.write(f'Predicted Leaf Class: {predictions[0]}')
```

Figure 4: Manual Input Implementation

Similar to the first section, there is a 'Predict' button that, when pressed, predicts and displays the leaf species based on the manually entered features (figure 4). The application also includes a feature to convert leaf class numbers into their corresponding scientific names. It loads a dataset from 'leaf.csv', which maps leaf class numbers to scientific names. Users enter a leaf class number, and upon pressing the 'Get Scientific Name' button, the application searches for the corresponding scientific name in the dataset and displays it. If the class number is not found, it notifies the user that the scientific name is not found (figure 5).

```
scientific_name = class_to_name.get(class_input, "Scientific name not found.")
```

Figure 5: Error Message implementation

4. SOFTWARE

4.1 Software Requirements

The "Leaf Identification" web application, as a Software Engineer, defining clear and comprehensive software requirements is crucial. These requirements encompass both functional and non-functional aspects to ensure the application meets user needs and performs effectively under various conditions.

2.1.1 Functional Requirements

Users must be able to input data via interactive sliders and input fields for each of the leaf features, such as Eccentricity, Aspect Ratio, Elongation, etc. The application should prevent invalid data entry by enforcing the specified ranges for each feature. Upon submission of the input data, the application should use the pre-loaded machine learning model to predict and display the leaf species. Predictions should be returned to the user in real-time or within a few seconds to ensure a responsive user experience. Users should have the capability to input a leaf class number and receive the corresponding scientific name, ensuring this feature supports all class numbers available in the dataset. The application should gracefully handle any errors during the input or prediction process and provide user-friendly error messages. Clear instructions should be provided to guide new users on how to input data and interpret results.

2.1.2 Non-Functional Requirements

The user interface should be intuitive, allowing users without technical expertise to easily navigate and use the application.The design should be clean and minimalistic, with a focus on usability and visual clarity. The system should be designed to scale easily, accommodating an increase in user numbers and data volume without significant changes to the infrastructure.

4.2 Software versioning

The project will be versioned using Git, with a private repository hosted on GitHub. The repository will be organized into branches, with each team member working on their own branch. The master branch will be used to merge the different components of the project, and the release branch will be used to deploy the final version of the project.

4.3 webapp development

A docker container will be used to deploy the web application. This will allow us to easily manage the different dependencies of the application and ensure its portability. A custom Docker image will be created using a Dockerfile, this will allow us to granularly define the different components of the application and ensure its reproducibility minimizing the software overhead.

4.4 Continuous Integration

A separate branch of the GitHub repository, release, will be used to publish the final version of the project after each sprint. Thanks to GitHub webhooks, the deployment of the new model will be automatically triggered after each push to the release branch, ensuring that the application is always up to date.

5. PROBLEMS ENCOUNTERED

Effective collaboration among software engineers, data engineers, and software developers is crucial when building complex applications like leaf classification tool. Each role brings unique skills and perspectives that are essential for addressing the various challenges and issues that can arise.

5.1 Complex Image Processing

Initially, the project aimed to utilize image processing techniques to extract features directly from leaf images for classification. This approach required sophisticated image processing skills and tools to accurately extract meaningful features from varied leaf images. The complexity of implementing effective image processing pipelines proved to be a hard so the project shifted focus from direct image processing to a user-input-based approach. This allowed the classification model to operate on manually entered leaf features rather than automatically extracted image data, simplifying the development process.

5.2 Model Prediction Issues

The model consistently predicted the same class for all inputs, which suggested issues with either the model itself or the way inputs were processed in the application. This issue was critical as it directly affected the core functionality of the application, rendering the classification tool ineffective. Extensive testing was conducted to isolate the issue, including testing the model with known data points and validating the integrity of the model file. The model retrained and problem solved.

5.3 Issues For Variable Ranges and Appropriate Error Messages

Users might input values that are out of the expected range for the features, which can lead to incorrect predictions or errors. I Clearly define and enforce input ranges for each variable using Streamlit's widget options such as min_value, max_value, and step. This ensures that all inputs are within the valid range the model expects.

6. DISCUSSION AND CONCLUSION

The Web Application's design is intentionally kept to a single page, which simplifies user interaction even for those without extensive training in its use. The ease of selecting the input method directly alongside input fields contributes to the application's straightforward functionality. During its use, I identified an area for improvement and added converting leaf class number to its scientific name. for this reason, I need to consult data engineer to check the feasibility. In this case I was also the data engineer so I added an extra column to database using documents related to database [1]. This enhancement significantly improves the user experience

7. TEAM STRUCTURE

In the ideal setup for this project on leaf identification using a web application, the workload was intended to be distributed among three key roles: a Data Engineer, a Software Engineer, and a Software Developer. Each role was designed to leverage specific skill sets to contribute to various aspects of the project, ensuring efficiency and specialization in handling complex tasks from data management to user interface design. Due to the unavailability of group mates for this project, I undertook all the responsibilities associated with each role. Managing these roles solo provided a unique challenge and learning opportunity, pushing me to acquire and apply a broad range of skills.

8. REFERENCES

[1] Silva,Pedro and Maral,Andr. (2014). Leaf. UCI Machine Learning Repository. https://doi.org/10.24432/C53C78.Ding, W. and Marchionini, G. 1997. A Study on Video Browsing Strategies. Technical Report. University of Maryland at College Park.

[2] Streamlit. (2022). Streamlit Documentation. Retrieved from https://docs.Streamlit.io