Architecture Design Document

Project Name: Mushroom Classification

Name: Venkata Phani Srikar Pillalamarri

E-mail: pvpsrikar@gmail.com

Table of Contents

Sr.No	Topic	Page No
1	**Introduction**	3
2	**Architecture**	4
2.1	Data Collection	4
2.2	Data Transformation	4
2.3	Data Preprocessing	5
2.4	Machine Learning Model	5
2.5	Flask Setup	6
2.7	Deployment	6
3	**Test Cases**	7
4	**Conclusion**	8

Chapter – 1

Introduction

The web application's low-level architectural diagram depicts the intricate interconnections among several components that are responsible for generating predictions using user input data. The backend of the design consists of many crucial components, including the Web Server, Model Loading Module, Preprocessing Module, and Prediction Module. Upon submission of data via the frontend user input form, the data is sent to the Web Server, which then coordinates the following processing procedures. The data is first sent to the Preprocessing Module, where it undergoes cleaning and transformation to conform to the specific format needed by the trained machine learning models. The processed data is next sent to the Prediction Module, which utilizes an ensemble model consisting of many trained Decision Tree Classifiers concatenated as a Random Forest Classifier, in order to provide predictions. The forecast outcomes are sent back to the Web Server and shown to the user via the frontend.

The deployment environment leverages a Flask server to host both the frontend and backend, ensuring seamless interaction between the user and the prediction system. The database component plays a crucial role in storing user inputs, prediction results, and logs, facilitating future analysis and performance monitoring. This architecture ensures that the application is robust, scalable, and capable of delivering accurate predictions efficiently. By detailing the interactions between each component, this low-level architecture diagram provides a clear roadmap for developers and stakeholders, ensuring that the application meets its intended functionality and performance criteria.

Chapter – 2

Architecture

Architecture Diagram:

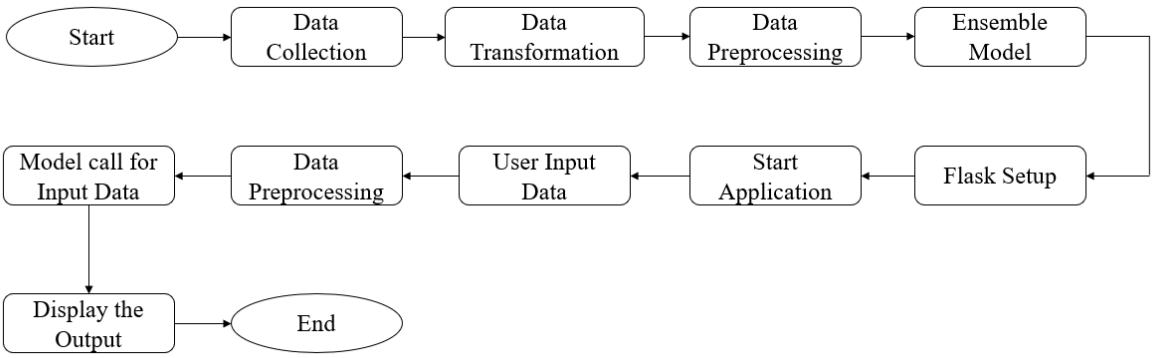

Fig: Architecture of the Design

2.1 Data Collection

Collecting data is an essential and pivotal stage in the construction of a machine learning model. During this stage, pertinent data is collected via Kaggle. The data must accurately reflect the issue domain and have all essential elements required for generating precise predictions. It is crucial to guarantee data quality at this stage, since the presence of noisy or missing data might have a negative impact on the performance of the model.

The data is then saved in a centralized repository, often a relational database or a cloud storage service, to enable convenient access and administration. Thorough documentation of the data sources, collecting techniques, and any preprocessing procedures used during collection is crucial. This documentation serves to preserve data provenance and guarantee the reproducibility or updating of the data in the future. In addition, it is essential to follow ethical guidelines and data privacy legislation while collecting data in order to safeguard sensitive information and meet legal requirements.

2.2 Data Transformation

After the collection of data, it is subjected to transformation in order to make it ready for analysis. Data transformation is the process of turning unprocessed data into a well-organized format that is appropriate for machine learning algorithms. This stage involves doing activities such as Label Encoding, which is the process of encoding the labels of the data. Data is often aggregated or blended from several sources throughout the process of transformation in order to get a complete dataset. This stage may entail addressing missing data by imputation using measures such as mean, median, or mode values, or using more sophisticated approaches such as interpolation. Anomalies are identified and resolved to avoid them from distorting the process of training the model. Data transformation is a procedure that guarantees the dataset is free from errors, maintains uniformity, and is presented in a way that improves the effectiveness and precision of the next data pretreatment and model training stages.

2.3 Data Preprocessing

Data preprocessing is an essential stage that comes after data transformation, with the purpose of getting the dataset ready for machine learning. This procedure involves doing activities like as data normalization, scaling, and dividing the dataset into training and testing sets. Normalisation guarantees that numerical characteristics are uniformly scaled, which is essential for algorithms that depend on distance metrics, such as K-Nearest Neighbours (KNN). To ensure that no one variable overwhelms the model's predictions, it is necessary to normalize characteristics such as the size of the mushroom cap and the intensity of its colour. Another crucial part of data preparation is managing categorical variables using encoding methods, such as one-hot encoding or label encoding. These approaches convert categorical data into a numerical representation. The dataset is often divided between training and testing sets to assess the model's performance on unfamiliar data. This division helps in evaluating the model's capacity to generalize. In addition, preprocessing involves doing feature engineering or selection procedures to improve the predictive capability of the dataset. This is achieved by developing new features or picking the most relevant ones.

2.4 Machine Learning Model/ Ensemble Model

The process of constructing machine learning models include the careful selection and training of algorithms capable of discerning patterns from data and generating predictions. Some often-used algorithms in machine learning include Logistic Regression, K-Nearest Neighbour (KNN), Naive Bayes, Support Vector Machine (SVM), Decision Tree, and Random Forests (RF). Each of these models has unique capabilities and is selected depending on the inherent nature of the issue and the specific features of the dataset. Logistic Regression is particularly advantageous for problems involving binary classification, while Decision Trees include the capability to handle both classification and regression tasks while also providing interpretability.

The training phase entails inputting the pre-processed training data into the chosen algorithms to acquire knowledge about the correlation between the input characteristics and the target variable. Model hyperparameters are adjusted using methods like as cross-validation to maximize performance. Subsequently, the trained models undergo evaluation using the testing set to quantify their accuracy, precision, recall, and other pertinent metrics. This assessment aids in identifying the most effective models, which may then be further improved or combined to maximize forecast accuracy.

An ensemble model is a technique that integrates numerous machine learning algorithms to enhance predicted performance by using the unique capabilities of each individual model. Ensemble models are often constructed using techniques like as bagging, boosting, and stacking. In the process of stacking, basic models such as Logistic Regression, KNN, Naive Bayes, SVM, Decision Tree, and RF are trained separately. Their predictions are then integrated using a meta-model, often a more intricate model like ANN, to get the ultimate prediction. The ensemble model seeks to mitigate overfitting and enhance resilience by aggregating the mistakes of separate models via averaging. By amalgamating forecasts from many models, the ensemble may attain superior accuracy and generalization compared to any one model in isolation. The ultimate ensemble model is assessed on the testing set to verify that it surpasses the performance of the individual models. This methodology often yields a more dependable

and precise forecasting system, making it a favoured option for several machine learning assignments.

2.5 Flask Setup

A Flask environment was established by setting up the necessary dependencies and configuring the application server. This involved installing Flask and any additional libraries required for handling HTTP requests and running the web server. An interactive user interface was developed using Flask's templating engine, Jinja2. The interface includes forms and dropdown menus for users to input various mushroom characteristics. The web pages are designed to be user-friendly and responsive, allowing users to easily provide data for classification.

2.6 Deployment

The last stage of the procedure involves deploying the web application, so becoming it accessible to end-users. This entails many processes, including installing the web server, establishing secure connections using SSL/TLS, and integrating with domain name services (DNS) to provide a user-friendly URL. The deployment procedure guarantees the accessibility, security, and optimal performance of the program across different scenarios. In addition, monitoring tools are established to monitor the performance of applications, user interactions, and the health of the system, enabling proactive maintenance and improvement. After the program has been deployed, it is crucial to consistently update it in order to address any issues, provide new features, and enhance current functionality. Automated deployment pipelines provide the smooth integration and distribution of updates, minimizing any interruptions. Obtaining user input at this stage is essential, as it allows for the identification of areas that need improvement and provides guidance for future development endeavours. In summary, deployment is a crucial stage that facilitates the transfer of the application from the development phase to the production phase, guaranteeing that it satisfies the requirements of the users and operates consistently in real-life situations.

Chapter – 3

Test Cases

Manual test cases are crucial for guaranteeing the quality and operation of a web application. Within the framework of the mushroom classifier web application, manual testing entails confirming that the application functions according to the user's expectations. Test cases are designed to include several facets of the program, such as input management, model prognostications, user interface components, and overall user encounter. The objective is to detect any potential problems or glitches that might impact the performance and user experience of the program before to its distribution to a broader user base.

Test Case – 1:

Features: ['x', 's', 'n', 't', 'p', 'f', 'c', 'n', 'k', 'e', 'e', 's', 's', 'w', 'w', 'p', 'w', 'o', 'p', 'k', 's', 'u']

Prediction Result: Poisonous

```
2024-08-03 21:45:49,491 - INFO - Home page accessed.
2024-08-04 12:11:47,999 - INFO - Model, encoders, and scaler loaded successfully.
2024-08-04 12:11:50,541 - INFO - Model, encoders, and scaler loaded successfully.
2024-08-04 12:11:51,600 - INFO - Home page accessed.
2024-08-04 12:19:44,516 - INFO - Received features: ['x', 's', 'n', 't', 'p', 'f', 'c', 'n', 'k', 'e', 'e', 's', 's', 'w', 'w', 'p', 'w',
'o', 'p', 'k', 's', 'u']
2024-08-04 12:19:44,546 - INFO - Prediction result: Poisonous
```

Fig: Log File for the above Features

Test Case – 2:

Features: ['x', 's', 'y', 't', 'a', 'f', 'c', 'b', 'k', 'e', 'c', 's', 's', 'w', 'w', 'p', 'w', 'o', 'p', 'n', 'n', 'g']

Prediction Result: Edible

```
2024-08-04 12:20:32,921 - INFO - Home page accessed.
2024-08-04 12:21:11,627 - INFO - Received features: ['x', 's', 'y', 't', 'a', 'f', 'c', 'b', 'k', 'e', 'c', 's', 's', 'w', 'w', 'p', 'w',
'o', 'p', 'n', 'n', 'g']
2024-08-04 12:21:11,646 - INFO - Prediction result: Edible
```

Fig: Log file for the above features

Test Case – 3:

Features: ['x', 'y', 'y', 't', 'a', 'f', 'c', 'b', 'n', 'e', 'c', 's', 's', 'w', 'w', 'p', 'w', 'o', 'p', 'k', 'n', 'g']

Prediction Result: Edible

```
2024-08-04 13:33:40,261 - INFO - Home page accessed.
2024-08-04 13:34:15,764 - INFO - Received features: ['x', 'y', 'y', 't', 'a', 'f', 'c', 'b', 'n', 'e', 'c', 's', 's', 'w', 'w', 'p', 'w',
'o', 'p', 'k', 'n', 'g']
2024-08-04 13:34:15,811 - INFO - Prediction result: Edible
```

Fig: Log file for the above features

Chapter – 4

Conclusion

The architectural diagram depicts the whole framework and sequence of operations of the mushroom classification web application. The procedure involves many phases, including data gathering, preprocessing, model training, ensemble model development, and deployment on Flask. Every element is carefully crafted to guarantee smooth integration and maximum efficiency of the program.

Test Cases:

Manual test cases play a crucial role in this design by verifying that every component of the application operates as intended. The test cases include several areas such as data input validation, model prediction accuracy, and user interface responsiveness. They provide a comprehensive framework for detecting and addressing possible faults. By including these test cases into the design, the mushroom classifier web application guarantees its dependability and user-friendliness. This results in a smooth user experience, from inputting data to receiving prediction results.

Architecture Description:

The architecture includes several key components:

1. Data Collection and Transformation: Gathering and structuring data for the model.
2. Data Preprocessing: Cleaning and preparing data for training.
3. Machine Learning Model/ Ensemble Model: Building Random Forest Classifier
4. Flask Setup and Deployment: Deploying the application for user access and interaction.

Every element is meticulously crafted to function together, guaranteeing streamlined data transmission and precise forecasts. The architecture is designed to be scalable and adaptable, enabling future improvements and the inclusion of further functionalities. The graphic offers a comprehensive perspective on the application's development, deployment, and maintenance processes by combining architectural description with rigorous test cases.