Architecture Design Document

Project Name: Mushroom Classification

Website Live Link: *Click Here*
Revision No: 1

Name: Chetan Mahale
Email: chetan.mahale0220@gmail.com

Table of Contents

Introduction

The low-level architecture diagram for the web application illustrates the detailed interactions between various components involved in making predictions based on user input data. At the core of the architecture is the backend, which encompasses several critical modules including the Web Server, Model Loading Module, Preprocessing Module, and Prediction Module. When a user submits data through the frontend user input form, the data is sent to the Web Server, which orchestrates the subsequent processing steps. The data is first passed to the Preprocessing Module, where it is cleaned and transformed to match the format required by the trained machine learning models. The transformed data is then sent to the Prediction Module, which leverages an ensemble model, comprising various trained models such as Logistic Regression, K-Nearest Neighbour, Naive Bayes, SVM, Decision Tree, and an Artificial Neural Network (ANN), to generate predictions. The prediction results are sent back to the Web Server and displayed to the user through the frontend.

The deployment environment leverages a Streamlit server to host both the frontend and backend, ensuring seamless interaction between the user and the prediction system. This environment is typically deployed on Streamlit's cloud platform. 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 1: Architecture

Architecture Diagram:

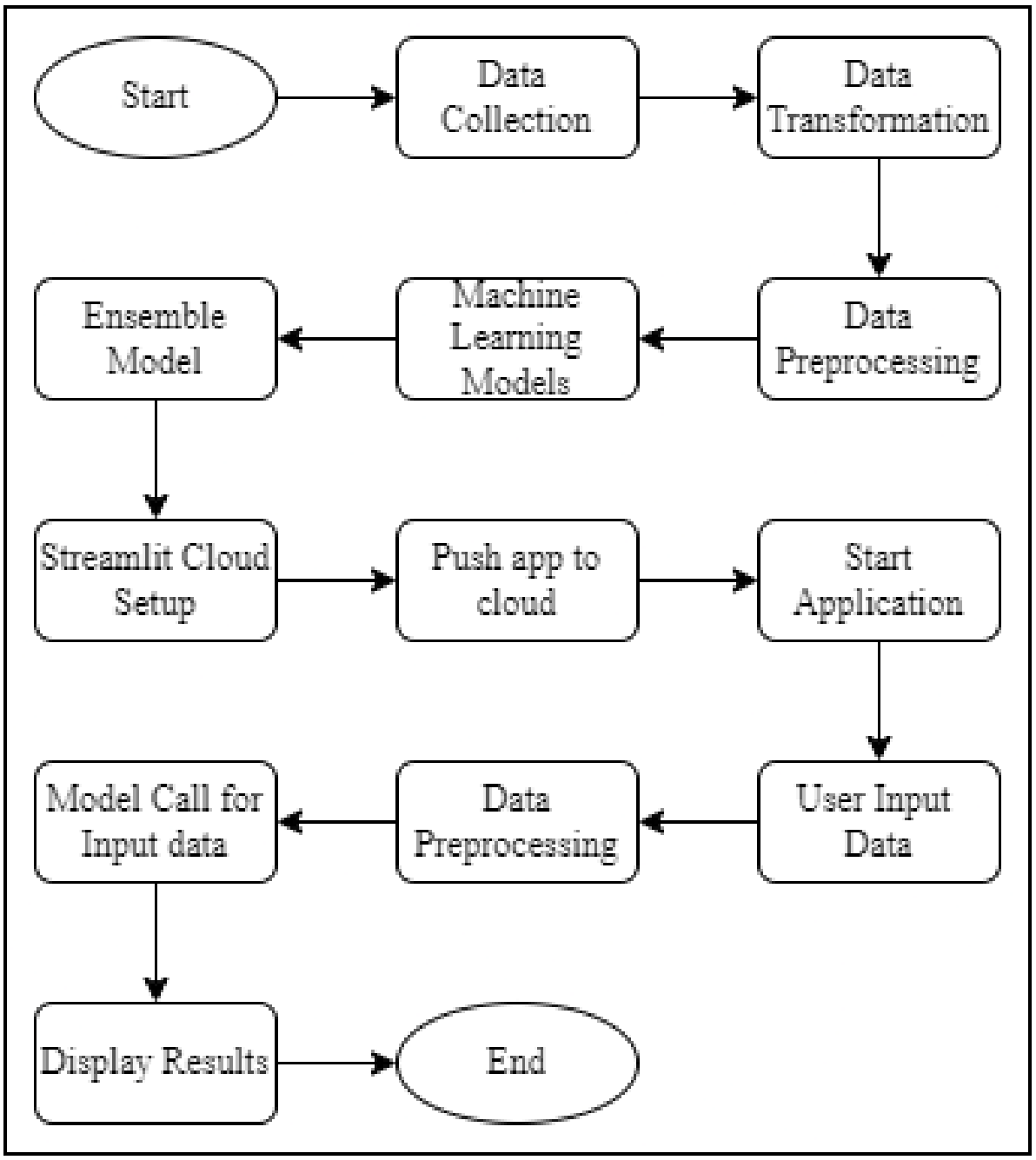

Fig: Architecture Diagram

1.1 Data Collection

Data collection is a crucial step in building a machine learning model. In this phase, relevant data is gathered from Kaggle. The data should be representative of the problem domain and include all the necessary features that will help in making accurate predictions. Ensuring data quality at this stage is essential, as noisy or incomplete data can adversely affect model performance.

The collected data is then stored in a centralised repository, often a relational database or a cloud storage service, to facilitate easy access and management. Proper documentation of the data sources, collection methods, and any preprocessing steps applied during collection is important. This documentation helps maintain data provenance and ensures that the data can be reproduced or updated in the future. Additionally, ethical considerations and data privacy regulations must be adhered to during data collection to protect sensitive information and comply with legal standards.

1.2 Data Transformation

Once data is collected, it undergoes transformation to prepare it for analysis. Data transformation involves converting raw data into a structured format suitable for machine learning algorithms. This step includes tasks such as Label Encoding, applying Principal Component Analysis(PCA) for dimensionality reduction.

During transformation, data is also often aggregated or merged from multiple sources to create a comprehensive dataset. This step may involve handling missing values by imputing them with mean, median, or mode values, or by using more advanced techniques like interpolation. Outliers are detected and addressed to prevent them from skewing the model training process. Data transformation ensures that the dataset is clean, consistent, and in a format that enhances the efficiency and accuracy of the subsequent data preprocessing and model training steps.

1.3 Data Preprocessing

Data preprocessing is a critical step that follows data transformation, aimed at preparing the dataset for machine learning. This process includes tasks such as data normalisation, scaling, and splitting the dataset into training and testing sets. Normalisation ensures that numerical features are on a similar scale, which is crucial for algorithms that rely on distance metrics, such as K-Nearest Neighbors (KNN). For instance, features like mushroom cap size and colour intensity need to be normalised to prevent any single feature from dominating the model's predictions.

Another important aspect of data preprocessing is handling categorical variables through encoding techniques, such as one-hot encoding or label encoding, which transform categorical data into a numerical format. The dataset is also typically split into training and testing sets to evaluate the model's performance on unseen data. This split helps in assessing the model's generalisation capability. Additionally, any feature engineering or selection processes are conducted during preprocessing to enhance the dataset's predictive power by creating new features or selecting the most relevant ones.

1.4 Machine Learning Models

Building machine learning models involves selecting and training algorithms that can learn patterns from the data and make predictions. Common algorithms include Logistic Regression, K-Nearest Neighbour (KNN), Naive Bayes, Support Vector Machine (SVM), Decision Tree, and Artificial Neural Networks (ANN). Each of these models has its strengths and is chosen based on the problem's nature and the dataset's characteristics. For example, Logistic Regression is useful for binary classification tasks, while Decision Trees can handle both classification and regression tasks with interpretability.

The training process involves feeding the preprocessed training data into the selected algorithms to learn the relationship between input features and the target variable. Model hyperparameters are tuned using techniques like cross-validation to optimise performance. The trained models are then evaluated on the testing set to measure their accuracy, precision, recall, and other relevant metrics. This evaluation helps identify the best-performing models, which can be further refined or ensembled to enhance predictive accuracy.

1.5 Ensemble Model

An ensemble model combines multiple machine learning algorithms to improve predictive performance by leveraging the strengths of each individual model. Techniques such as bagging, boosting, and stacking are commonly used to create ensemble models. In stacking, for example, base models like Logistic Regression, KNN, Naive Bayes, SVM, Decision Tree, and ANN are trained independently, and their predictions are combined using a meta-model, often a more complex model like ANN, to make the final prediction.

The ensemble model aims to reduce overfitting and improve robustness by averaging out the errors of individual models. By combining predictions from diverse models, the ensemble can achieve higher accuracy and generalisation than any single model alone. The final ensemble model is evaluated on the testing set to ensure it outperforms the individual models. This approach often results in a more reliable and accurate prediction system, making it a preferred choice for many machine learning tasks.

1.6 Streamlit Cloud setup

Setting up Streamlit Cloud involves deploying the web application to a cloud platform to make it accessible to users over the internet. Streamlit is a popular open-source framework for creating interactive web applications for machine learning and data science. The deployment process starts with setting up a Streamlit account and configuring the deployment environment.

Once the environment is configured, the web application, along with the trained machine learning models and any necessary dependencies, is deployed to the cloud. The deployment involves pushing the code repository to the cloud platform, configuring environment variables, and setting up continuous integration and continuous deployment (CI/CD) pipelines for automated updates. Streamlit Cloud also provides features for managing user access, monitoring application performance, and scaling resources to handle increased traffic, ensuring a smooth and responsive user experience.

1.7 Deployment

The final step of the process is deploying the web application, making it live for end-users. This involves several tasks, such as configuring the web server, setting up secure connections using SSL/TLS, and integrating with domain name services (DNS) to provide a user-friendly URL. The deployment process ensures that the application is accessible, secure, and performs well under various conditions. Additionally, monitoring tools are set up to track application performance, user interactions, and system health, allowing for proactive maintenance and optimization.

Post-deployment, it's essential to regularly update the application to fix bugs, add new features, and improve existing functionalities. Automated deployment pipelines ensure that updates are seamlessly integrated and deployed with minimal downtime. User feedback is also crucial at this stage, as it helps identify areas for improvement and guides future development efforts. Overall, deployment is a critical phase that transitions the application from development to production, ensuring it meets the users' needs and performs reliably in real-world scenarios.

Chapter 2: Test Cases

Manual test cases are essential for ensuring the quality and functionality of a web application. In the context of the mushroom classifier *web application*, manual testing involves verifying that the application performs as expected from a user's perspective. Test cases are designed to cover various aspects of the application, including input handling, model predictions, user interface elements, and overall user experience. The goal is to identify any issues or bugs that may affect the application's performance and usability before it is released to a wider audience.

Test Case Id	Test Case Description	Pre-Conditions	Test Steps	Expected Result	Status
TC01	Verify the home page loads correctly	User is on the mushroom classifier URL	1. Navigate to Website 2. Observe the home page	The home page should load without errors, displaying the application title and input form for mushroom classification	Pass
TC02	Verify data input fields are present	User is on the home page	1. Observe the input form on the home page	Input fields for all necessary attributes should be present and labelled correctly.	Pass
TC03	Verify user can input data into fields	User is on the home page	1. Enter valid data into each input field	Users should be able to enter data into each field without any errors.	Pass
TC04	Verify 'Classify' button functionality	User has entered valid data	1. Click the 'Classify' button	The application should process the input data and display the prediction result on the screen.	Pass
TC05	Verify model prediction accuracy	User has entered valid data	1. Enter data corresponding to known mushroom types. 2. Click the 'Classify' button	The application should display the correct prediction (edible or poisonous) based on the input data.	Pass
TC06	Verify responsiveness of the web	User is on the home page	1. Resize the browser window to various sizes	The application should be responsive and	Pass

	application		(mobile, tablet, desktop)	adjust its layout to fit different screen sizes without any loss of functionality or readability.	
TC07	Verify overall user experience	User uses the application	1. Use the application to make several predictions 2. Navigate through the application features	The overall user experience should be smooth, intuitive, and free of any bugs or issues that hinder the user from making predictions and navigating the application.	Pass

Conclusion

The architecture diagram illustrates the comprehensive structure and workflow of the mushroom classifier web application. It encompasses various stages from data collection, preprocessing, model training, and ensemble model creation to deployment on Streamlit Cloud. Each component is meticulously designed to ensure seamless interaction and optimal performance of the application.

Test Cases:
Manual test cases are integral to this architecture, ensuring each element of the application functions as intended. These test cases cover aspects like data input validation, model prediction accuracy, and user interface responsiveness, providing a robust framework for identifying and resolving potential issues. The integration of these test cases into the architecture ensures the reliability and usability of the mushroom classifier web application, offering a seamless user experience from data input to 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 Models: Building individual models such as logistic regression, KNN, Gaussian NB, SVM, decision tree, and ANN.
4. Ensemble Model: Combining individual models into a robust ensemble for improved accuracy.
5. Streamlit Cloud Setup and Deployment: Deploying the application for user access and interaction.

Each component is designed to work cohesively, ensuring efficient data flow and accurate predictions. The architecture supports scalability and adaptability, allowing for future enhancements and integration of additional features. By incorporating both architecture description and rigorous test cases, the diagram provides a holistic view of the application's development, deployment, and maintenance processes.