**Low-Level Design**

**Mushroom Classification**

**History:**

| **Version** | **Date** | **Author** | **Comments** |
| --- | --- | --- | --- |
| 1.0 | 17-Jan-25 | Ankit Sharma | Initial LLD document for mushroom project |
|  |  |  |  |

**Approval Status:**

| **Version** | **Review Date** | **Reviewed By** | **Approved By** | **Comments** |
| --- | --- | --- | --- | --- |
| 1.0 |  |  |  |  |
|  |  |  |  |  |

**Content**

1. Introduction
   * 1.1 What is a Low-Level Design Document?
   * 1.2 Scope
2. Architecture
3. Architecture Description
   * 3.1 Data Transformation
   * 3.2 Data Preprocessing
   * 3.3 Model Building
   * 3.4 API Deployment
4. Unit Test Cases
   * 4.1 Test Cases for Data Preprocessing
   * 4.2 Test Cases for Model Prediction
   * 4.3 Test Cases for API Endpoints
5. Deployment Details
6. Conclusion

**1. Introduction**

**1.1 What is a Low-Level Design Document?**

A Low-Level Design (LLD) document provides the internal logical design of the application, detailing the program modules, class diagrams, methods, and relationships. It acts as a blueprint for developers to implement the project, covering essential details like data preprocessing, model building, and API functionality.

**1.2 Scope**

This LLD focuses on the mushroom classification project, detailing:

* Data transformation and preprocessing steps
* Machine learning pipeline and hyperparameter tuning
* Model deployment on Render using Flask API
* Unit testing of individual components

**2. Architecture**

The architecture of the mushroom classification system is modular, involving the following components:

1. **Data Preprocessing**: Handles missing values and encodes categorical features.
2. **Model Training**: Implements machine learning pipelines and hyperparameter tuning.
3. **Model Deployment**: Deploys the model as an API using Flask.
4. **Prediction and Logging**: Processes user input and logs requests and predictions.

**3. Architecture Description**

**3.1 Data Transformation**

* **Tools**: Scikit-learn (LabelEncoder, SimpleImputer)
* **Steps**:
  1. Encode all categorical features using LabelEncoder.
  2. Handle missing values with SimpleImputer (strategy: most frequent).

**3.2 Data Preprocessing**

* Combines transformations using Scikit-learn’s Pipeline to manage categorical data efficiently.
* Ensures all categorical columns are encoded and any missing values are handled appropriately.

**3.3 Model Building**

* **Steps**:
  1. Train multiple models (e.g., Logistic Regression, Random Forest, Gradient Boosting).
  2. Use Grid Search for hyperparameter tuning.
  3. Evaluate models using metrics like accuracy, precision, recall, and F1-score.
* **Output**: Save the best model using pickel for deployment.

**3.4 API Deployment**

* **Framework**: Flask
* **Deployment Platform**: Render
* **Endpoints**:
  + /: Accepts JSON input and returns predictions (e.g., "edible" or "poisonous").
  + /predict: Returns the status of the API.
* **Logging**: Use Python’s logging library for tracking requests and debugging errors.

**4. Unit Test Cases**

**4.1 Test Cases for Data Preprocessing**

| | **Test Case Description** | **Pre-Requisite** | **Expected Result** | | --- | --- | --- | | Verify handling of missing values | Input dataset with missing values | Returns dataset with imputed values | | Verify encoding of categorical variables | Input dataset with categorical data | Returns dataset with encoded variables |   **4.2 Test Cases for Model Prediction** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| | **Test Case Description** | **Pre-Requisite** | **Expected Result** | | --- | --- | --- | | Verify model predicts edible or poisonous | Input sample data | Prediction: "mushroom is poisonous" or "edible mushroom" | | Verify prediction response time | Deployed API | Response time < 500ms |   **4.3 Test Cases for API Endpoints** |
| | **Test Case Description** | **Pre-Requisite** | **Expected Result** | | --- | --- | --- | | Verify API accessibility | API deployed on Render | Status code: 200 | | Verify / endpoint functionality | Valid JSON input to /predict | Returns prediction in JSON format | | Verify /predict endpoint functionality | API deployed on Render | Returns "API is predict" | |

**5. Deployment Details**

* **Platform**: Render
* **Framework**: Flask
* **Model Format**: pickle serialized file
* **API Endpoints**:
  + **/:** Accepts input and provides predictions.
  + **/predict:** Verifies API health status.
  + **/train:** train the model

**6. Conclusion**

This document provides a detailed low-level design for the mushroom classification project, covering the essential components from data preprocessing to deployment and testing. It ensures that developers can implement, test, and deploy the solution efficiently, maintaining high reliability and accuracy.