Low Level Design (LLD)

Mushroom Classification Application

By

VIJAI VIKRAM I

Revision Number: 1.0
Last date of revision: 10/04/2024

Document Version Control

Date Issued	Version	Description Author
10/04/2024	1	Initial LLD - V1.0 Vijai Vikram

Contents

Abstract

The Low-Level Design (LLD) document for the Mushroom Classification machine learning project provides a comprehensive guide to the intricacies of the system's architecture and implementation. Commencing with a clear introduction, it outlines the purpose and scope, emphasizing its role in detailing design decisions and specifications for the Mushroom Classification system. The document illustrates the system architecture, showcasing relationships and interactions between components. It meticulously explains the chosen classification algorithm, its architecture, and the necessary input features, offering insights into model training methodologies and validation techniques. API design considerations cover endpoints, input validation, error handling, and security measures. Deployment strategies, including environment specifications and containerization details, are outlined for seamless implementation. Testing methodologies, encompassing both unit and integration testing, ensure the reliability of the machine learning model and API functionalities. Loggingmechanisms and monitoring metrics are defined for effective debugging and system health assessment. Security measures address data privacy concerns, outlining strategies to safeguard sensitive data. Additionally, model explainability techniques are incorporated to enhance transparency in predictions. This abstract serves as a succinct reference for stakeholders involved in the development and evolution of the Mushroom Classification system.

1 Introduction

1.1 Why this Low-Level Design Document?

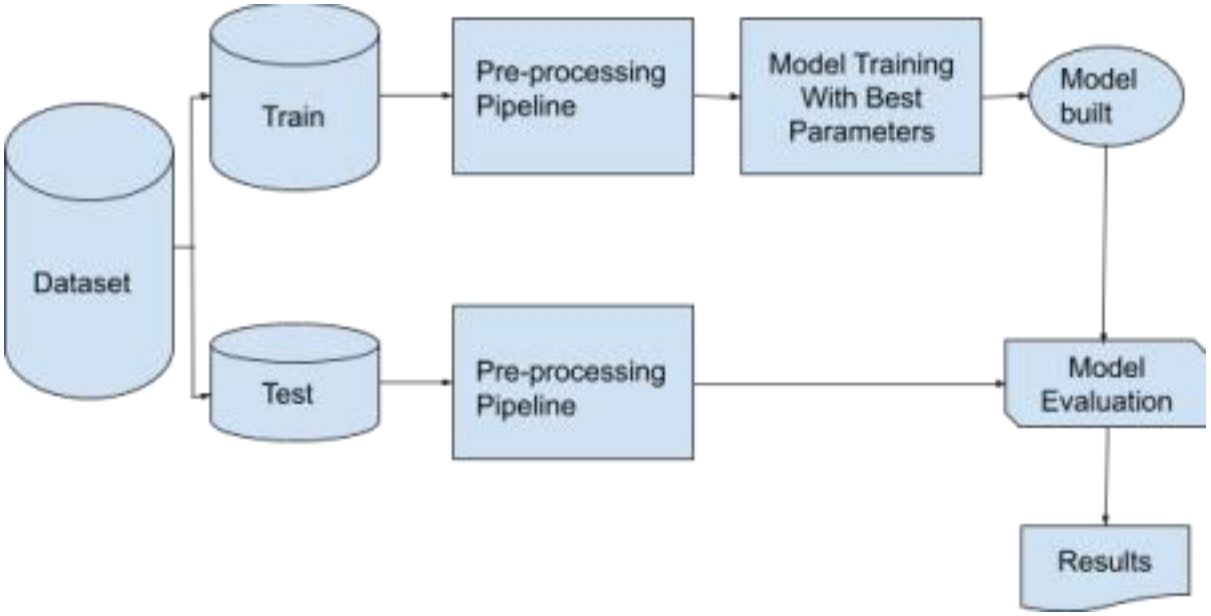
The Low-Level Design (LLD) document for the Mushroom Classification system is crafted tooffer a detailed portrayal of the system's inner workings. It aims to provide an in-depth understanding of the system's architecture, features, interfaces, operational constraints, and responses to external stimuli. This document is crucial for both stakeholders and developers, serving as a proposal for approval from higher management. It extensively describes the internal design of the application, focusing on the interactions and functionalities of various components.

1.2 Scope

Low-level design (LLD) represents a component-level design process characterized by a step-by-step refinement. This process is instrumental in designing data structures, establishing software architecture, and developing source code. The overall organization of data is initially defined during requirement analysis and undergoes further refinement during data design work. This section delineates the scope of the LLD process for the Mushroom Classification system, providing a foundation for

subsequent design details.

2 System Architecture



3 Machine Learning Components

3.1.1 Data Gathering

The project utilizes a dataset sourced from Kaggle, accessible through the following link: Mushroom Classification Dataset.

3.1.2. Tools Used

- Programming Language: Python 3.8, with frameworks such as numpy, pandas,scikit-learn, and other relevant modules for model development.
- Integrated Development Environment (IDE): Vscode.
- Visualization Tools: Seaborn and components of Matplotlib.
- Version Control: GitHub for code versioning.
- Deployment: Netlify is employed for project deployment.

3.1.3. Data Description

The Mushroom dataset is a comprehensive dataset with 8314 rows and 23 columns.

3.2 Preprocessing Pipeline

The Preprocessing Pipeline prepares the dataset for model training by handlingmissing values, encoding categorical variables, and scaling features.

Imputation: Define the strategy for handling missing values. **Encoding:**

Specify the encoding method for categorical variables.**Scaling:** Describe the method used for feature scaling.

- Cleaning and transforming raw data into a format suitable for model training.
- Ensuring data consistency and compatibility with the chosen machine learning algorithm

3.3 Model Training & Hyper-parameter Tuning

The Model Training & Hyper-parameter Tuning Component focuses on training amachine learning model to predict credit card defaults.

Algorithm Selection: Specify the chosen classification algorithm (e.g., RandomForest, Logistic Regression).
Hyper-parameter Tuning: Define the hyper-parameters to be tuned.
Cross-validation: Describe the cross-validation strategy.

- Training the machine learning model on the preprocessed dataset.
- Fine-tuning hyper-parameters to enhance model performance.

3.4 Model Evaluation

The Model Evaluation Component assesses the performance of the trained model.

Metrics: Define evaluation metrics (e.g., accuracy, precision, recall, F1-score).**Cross-validation Evaluation**: Specify the strategy for cross-validated model evaluation.

- Evaluating model performance on both training and validation sets.
- Calculating and reporting relevant metrics to assess prediction quality.

4 API Design

4.1 Front End Application

Develop a front end application using Flask for API and HTML as user-interface toaccept input data to the model

4.2 API Endpoints

Expose API endpoints for the model to consume requests from users and revertthe prediction as an output.

5 Model Deployment

5.1 AWS

3. Deployment on AWS EC2 Instance

1. Prepare Your Model:

- Ensure your machine learning model is trained and ready for deployment.
- Save or export your model in a format compatible with your chosen framework.

2. Docker Containerization (Using ECR):

- Containerize your application using Docker.
- Create a Docker image and push it to AWS Elastic Container Registry (ECR).
- Ensure your Docker image contains the necessary dependencies, model files, and application code.

3. IAM Roles:

- Utilize IAM roles to manage permissions securely.
- Define IAM roles with the necessary permissions for your EC2 instance to accessother AWS services like ECR.

4. Launch EC2 Instance:

- Launch an EC2 instance and associate it with the IAM role granting necessary permissions.
- Pull the Docker image from ECR to the EC2 instance.

5. Security Measures:

- Implement security measures for your EC2 instance.
- Configure security groups and network settings to control incoming and outgoingtraffic.

6. Monitoring and Logging:

- Utilize AWS CloudWatch for monitoring your deployed EC2 instance.
- Configure logging within your application to capture relevant information.
- Set up CloudWatch Alarms for automated notifications based on defined metrics.

By adopting these steps, you can deploy your machine learning model on an AWS EC2 instance using IAM, ECR, and Docker containerization for a secure and scalable deployment environment.

6 Testing

6.1 Unit Testing

Unit testing is crucial to ensure the correctness of individual components in the mushroom classification system.

6.1.1 Unit Test Cases

Test Case Description	Pre-Requisites	Expected Results
Verify whether the User Interface URL is accessible to the user.	1. User Interface URL should be defined.	User Interface URL should be accessible to the user.
Verify whether the User Interface loads completely for the user when the URL is accessed.	1. User Interface URL is accessible. 2. User Interface is deployed.	The User Interface should load completely for the user when the URL is accessed.
Verify whether user is able to edit all input fields.	1. User Interface is accessible.	User should be able to edit all input fields.

7 Conclusion

This Low-Level Design document provides a comprehensive overview of the design intricacies of the mushroom classification machine learning project. Each component discussed within the document is meticulously crafted to fulfill specific roles, contributing collectively to a robust and efficient mushroom classification solution.

The design decisions and specifications outlined here form the foundation for the successful implementation of the system. From data gathering and preprocessing to model training, evaluation, and deployment, every aspect has been carefully considered to ensure the accuracy and reliability of the classification model.

By utilizing tools such as Python, seaborn, and others, along with leveraging machine learning algorithms and frameworks, the project aims to achieve optimal performance in different scenarios. The integration of Cassandra for data storage and retrieval, coupled with API development for seamless interaction, adds to the versatility and scalability of the system.

The deployment on AWS, utilizing services such as EC2 and ECR, signifies a commitment to providing users with easy access to the mushroom classification system. Security measures,including IAM roles and encryption, are implemented to safeguard both the model and user data.

Documentation, including code documentation generated with Sphinx and a user guide, ensures clarity for both developers and end-users. Unit testing procedures are established to validate the correctness of critical functions, enhancing the overall robustness of the system.

In conclusion, this Low-Level Design document serves as a roadmap for the development and evolution of the mushroom classification system, aiming to deliver accurate predictions while maintaining transparency and security. It lays the groundwork for a solution that not only meets the current requirements but is also adaptable to future enhancements and advancements in the field of machine learning and classification.