Low-Level Design (LLD)

Mushroom Classification

Written By	Ayush Wase
Document Version	LLD - V.1.1
Last Revised Date	03-June-2023

Document Version Control

Version	Date	Author	Comments
HLD - V.1.0	02-June-2023	Ayush Wase	Initial LLD - V.1.0
HLD - V.1.1	26-May-2023	Ayush Wase	Document updated

Contents

ABSTRACT

This study aimed to classify mushrooms using different machine learning models, including Adaboost, KNN, QDA, Random Forest, SVM, XGBoost, and decision tree. The dataset comprised diverse mushroom samples with class labels. Performance evaluation metrics such as accuracy, precision, recall, and F1-score were used to compare the models. The decision tree algorithm emerged as the top performer, surpassing the other models in all evaluation metrics. Its success can be attributed to its hierarchical decision-making structure based on feature thresholds, which enables efficient classification. Moreover, decision trees provide interpretable rules for better understanding. These findings underscore the importance of selecting appropriate algorithms for mushroom classification, with decision trees showing promise for accurate classification based on mushroom attributes.

1. Introduction

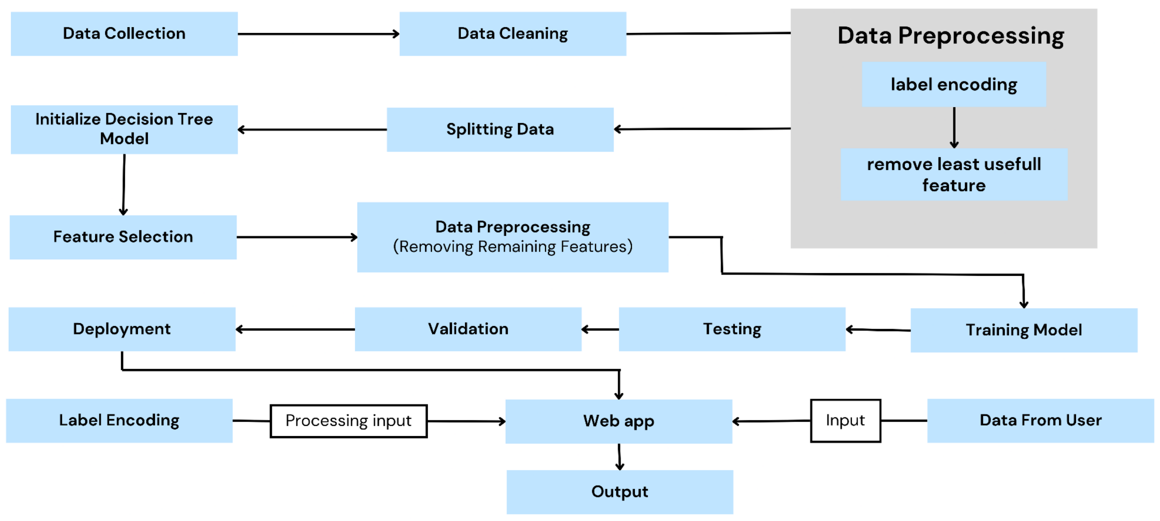
1.1 Why this Low-Level Design Document?

This Low-Level Design Document (LLD) serves as a comprehensive guide to the technical implementation details of the mushroom classification system. It provides a detailed description of the design decisions, algorithms, and methodologies used in developing the system. The LLD is crucial for ensuring that the development team has a clear understanding of the system's architecture, components, and their interactions. It also facilitates collaboration among team members and serves as a reference for future maintenance and enhancements of the system. By documenting the low-level design, this document helps maintain consistency, reduces ambiguity, and ensures efficient development and implementation of the mushroom classification system.

1.2 Scope

The scope of this mushroom classification system is to accurately classify mushrooms based on their attributes using machine learning algorithms. The system aims to analyze a diverse dataset of mushroom samples, each labeled with its corresponding class. It covers a range of machine learning models, including Adaboost, KNN, QDA, Random Forest, SVM, XGBoost, and decision tree. The system focuses on evaluating the performance of these models using various metrics such as accuracy, precision, recall, and F1-score. The scope also includes comparing and identifying the most effective algorithm for mushroom classification. The system is designed to provide insights into the attributes that contribute to accurate classification and aid in understanding the decision-making process. The classification system's primary goal is to enhance the accuracy and efficiency of mushroom classification, ultimately benefiting researchers, enthusiasts, and industries working with mushrooms.

2. Architecture



3. Architecture Description

1. Data Collection:

The dataset used in this study contains descriptions of hypothetical samples of gilled mushrooms from 23 species. These species belong to the Agaricus and Lepiota Family Mushroom and were drawn from The Audubon Society Field Guide to North American Mushrooms (1981). Each species is categorized as definitely edible, definitely poisonous, or of unknown edibility and not recommended (combined with the poisonous class). The dataset provides information on various attributes of the mushrooms, such as cap shape, cap surface, cap color, bruises, odor, gill attachment, gill spacing, gill size, gill color, stalk shape, stalk root, stalk surface above and below ring, stalk color above and below ring, veil type and color, ring number and type, spore print color, population, and habitat. The aim of this study is to develop a model that can accurately classify mushrooms based on these attributes, enabling identification of edible and poisonous varieties.

2. Data Cleaning:

In the initial step, the necessary libraries, such as NumPy, pandas, and sklearn, are imported. The dataset is loaded using the `**read_csv()**` function from pandas, which reads the data from the specified file **("/content/mushrooms.csv")**. The **info()** method is then used to obtain an overview of the dataset. From the output, we can see that the dataset contains 8,124 entries with 23 columns, all of which are of object type.

3. Data Preprocessing:

To preprocess the data and make it suitable for analysis, the categorical variables in the dataset need to be converted into a numeric form. The label encoder from sklearn's preprocessing module is used to achieve this. A loop is implemented to iterate over each column in the dataset, and the label encoder is applied to convert the categorical values into numeric labels. This process ensures that the data is in a suitable format for further analysis. The encoded data is then stored back into the respective columns of the dataframe. The **"veil-type"** column is dropped using the **drop()** function, as it contains only a single unique value and does not contribute to the classification task.

4. Splitting Data:

The dataset is split into features and labels. The target variable, **"class"** is assigned to the variable **y**, while the remaining columns are assigned to the variable **X**. The **train_test_split()** function from sklearn's model_selection module is used to split the data into training and testing sets. In this case, the test size is set to 0.2, indicating that 20% of the data will be used for testing, while the remaining 80% will be used for training. Additionally, the training set is further split into training and validation sets using the same function. The test size for this second split is set to 0.25, resulting in a 60:20:20 ratio for training, validation, and testing, respectively.

5. Initialize Decision Tree Model:

An instance of the DecisionTreeClassifier model from sklearn's tree module is created and assigned to the variable **model**. This model will be used to build a decision tree classifier. The decision tree algorithm is a popular choice for classification tasks as it creates a tree-like model of decisions and their possible consequences.

6. **Feature Selection:**

To identify the most relevant features in the dataset, Recursive Feature Elimination (RFE) is applied. RFE is a feature selection technique that works by recursively eliminating less significant features from the dataset. It starts by fitting the model (DecisionTreeClassifier in this case) on the training data and then eliminates the least important features based on their rankings. This process continues iteratively until the desired number of features is selected. In this code, the **RFE()** function from sklearn's feature_selection module is used with the **model** (DecisionTreeClassifier) as the estimator and **n_features_to_select** set to 5, indicating that the aim is to select the top 5 features. After applying RFE, the indices of the selected features are obtained using the **get_support(indices=True)** method, and their corresponding names are extracted from the original dataset's columns using the **columns** attribute. The selected features are: **'gill-size', 'gill-color', 'stalk-root', 'spore-print-color', and 'population'.**

7. Data Preprocessing (Removing Rest of Features):

To focus on the selected features, the training, validation, and testing datasets are filtered to include only those features. This is done by assigning the selected feature names to the respective datasets: **X_train_selected**, **X_val_selected**, and **X_test_selected**. These datasets will be used for training, validation, and testing the model, respectively.

8. Training Model:

The decision tree model (**model**) is trained using the **fit()** method. The training data consists of the selected features (**X_train_selected**) and their corresponding labels (**y_train**). By fitting the model on the training data, it learns the patterns and relationships between the selected features and their associated labels.

9. Testing:

The trained decision tree model is then used to make predictions on the validation dataset (**X_val_selected**). The predictions are compared to the actual labels (**y_val**) to evaluate the model's performance. The **classification_report()** function from sklearn's metrics module is used to generate a comprehensive report on the classification results. This report includes metrics such as precision, recall, F1-score, and support for each class (0 and 1).

10. Validation:

The validation results show that the decision tree model performs well on the validation dataset. The accuracy of the model is calculated as 0.99, indicating that it correctly predicts the class labels for 99% of the validation instances. The precision, recall, and F1-score are high for both classes (0 and 1), which suggests that the model can effectively distinguish between the two classes. The macro average and weighted average of these metrics are also close to 0.99, indicating a balanced performance across the classes. The support values represent the number of instances for each class in the validation set.

11. Deployment:

The decision tree model has been deployed on a web application, allowing users to interact with the model and make predictions using the selected features. The web application link is provided at the bottom. By clicking on the link, users will be directed to the web app interface, where they can input the values for the selected features (**gill-size, gill-color, stalk-root, spore-print-color, and population**). Once the user submits the input, the deployed model will use the provided values to make predictions and display the predicted class label. The web app provides a user-friendly interface, allowing users to easily interact with the model and obtain predictions for mushroom classification based on the selected features. This deployment enables users to utilize the model's capabilities without requiring any programming or technical knowledge.

4. Unit Test Cases

Test Case	Prerequisites	Expected Result
Test Case 1	The user selects gill-size=b, gill-color=k, stalk-root=b, spore-print-color=k, population=a, and presses submit.	The web app should print "The mushroom is Edible."
Test Case 2	The user selects gill-size=n, gillcolor=r, stalk-root=c, spore print-color=n, population=v, and presses submit	The web app should print "The mushroom is Poisonous.
Test Case 3	The user selects gill-size=n, gill-color=b, stalk-root=u, spore-print-color=w, population=c, and presses submit	The web app should print "The mushroom is Edible."
Test Case 4	The user selects gill-size=b, gill-color=h, stalk-root=e, spore-print-color=h, population=s, and presses submit	The web app should print "The mushroom is Poisonous."
Test Case 5	The user selects gill-size=b, gill-color=y, stalk-root=z, spore-print-color=u, population=n, and presses submit	The web app should print "The mushroom is Edible."
Test Case 6	Open the web app's URL in a web browser	The web app should open successfully and display the homepage with the title "Mushroom Characteristics"
Test Case 7	Click on the "Predict" button without selecting any options	The web app should remain on the same page and not render the result page
Test Case 8	Select valid options for all the dropdown menus and press submit	The web app should render the result page with the appropriate prediction ("Edible" or "Poisonous") based on the selected options
Test Case 9	Select a valid option for all dropdown menus except one and press submit	The web app should remain on the same page and not render the result page, indicating that all options

		need to be selected
Test Case 10	Enter the URL of the result page directly in the web browser	The web app should not display the result page and should redirect to the home page
Test Case 11	Refresh the result page	The web app should reload the result page and display the previously predicted result
Test Case 12	Close the web app and reopen it	The web app should open successfully without any errors or issues