LOW-LEVEL DOCUMENT

Thyroid Predictor

Contents

1. Introduction

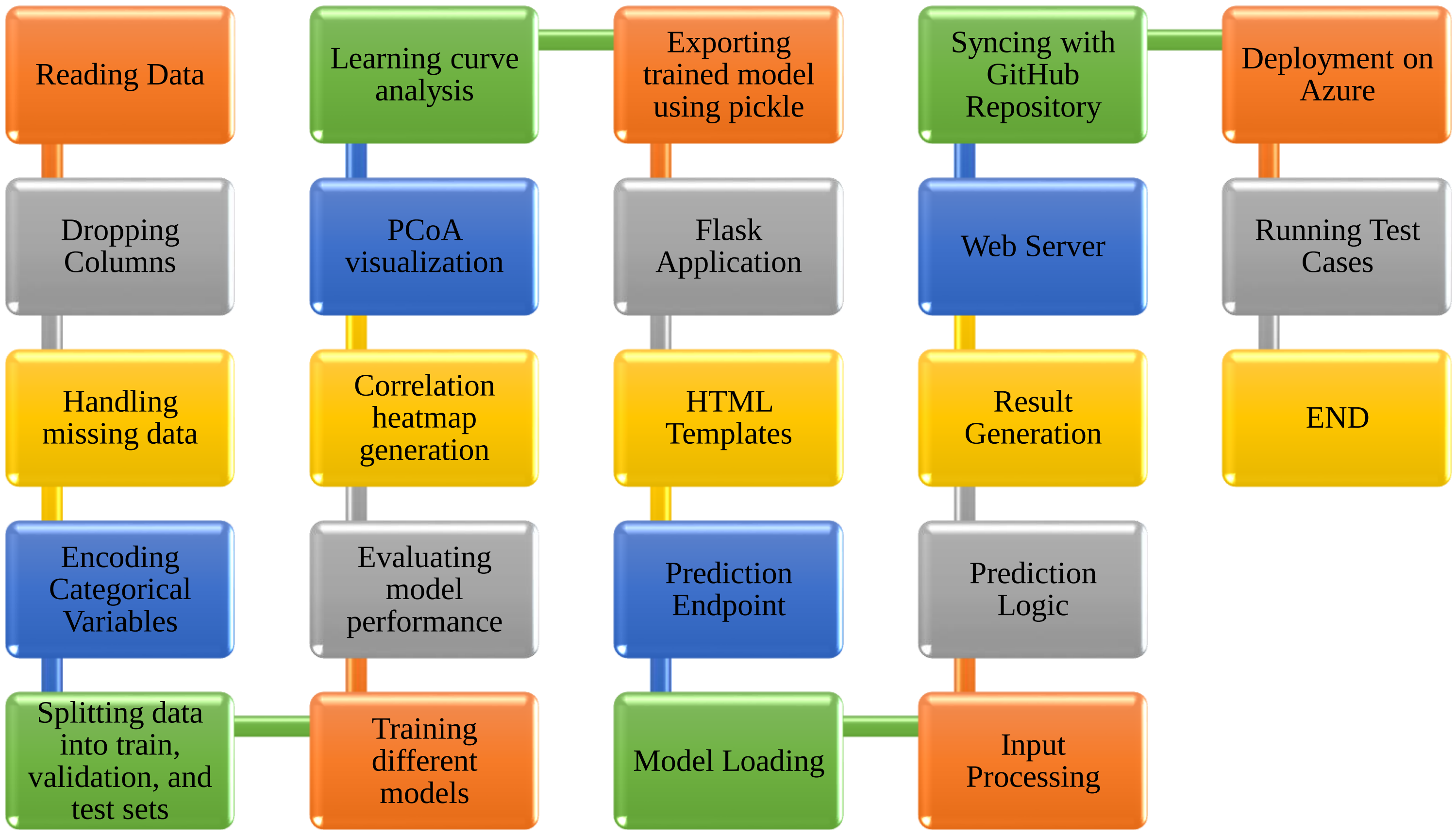
1.1. What is Low-Level design document?

The goal of LLD or a low-level design document (LLDD) is to give the internal logical design of the actual program code for Food Recommendation System. LLD describes the class diagrams with the methods and relations between classes and program specs. It describes the modules so that the programmer can directly code the program from the document.

1.2. Scope

Low-level design (LLD) is a component-level design process that follows a step-by-step refinement process. This process can be used for designing data structures, required software architecture, source code and ultimately, performance algorithms. Overall, the data organization may be defined during requirement analysis and then refined during data design work.

2. Architecture



3. Architecture Description

3.1 Data Preprocessing:

3.1.1 Data Loading:

- Utilizes the pandas library to read the dataset (`thyroid_dataset.csv`).

- Checks for any inconsistencies in the dataset such as missing values, duplicate entries, or erroneous data types.

3.1.2 Data Cleaning:

- Drops columns that are not relevant to the prediction task or contain a large number of missing values.

- Handles missing values by either removing corresponding rows or imputing values based on the nature of the missingness and the feature distribution.

- Performs any necessary transformations on the data, such as normalization or scaling, to ensure better model performance.

3.1.3 Feature Engineering:

- Encodes categorical variables using appropriate techniques such as one-hot encoding or label encoding.

- Considers domain knowledge to engineer new features or transform existing ones to improve the predictive power of the model.

3.1.4 Data Visualization:

- Utilizes seaborn and matplotlib libraries to create visualizations such as histograms, boxplots, and correlation matrices to gain insights into the data distribution, identify outliers, and understand relationships between variables.

- Visualizations aid in making informed decisions during data preprocessing and feature engineering stages.

3.2 Model Training:

3.2.1 Data Splitting:

- Splits the preprocessed data into training, validation, and test sets using the `train_test_split` function.
- Allocates a portion of the data for training the model, another portion for tuning hyperparameters and model selection (validation set), and the remaining portion for evaluating the final model performance (test set).

3.2.2 Model Selection and Training:

- Instantiates various machine learning models (e.g., Logistic Regression, Decision Tree, Random Forest, KNN, SVM) using scikit-learn.
- Trains each model on the training set using the `fit` method, allowing the model to learn patterns from the data.

3.2.3 Model Evaluation:

- Evaluates the performance of each trained model on the validation set using appropriate evaluation metrics such as accuracy, precision, recall, and F1-score.
- Compares the performance of different models to select the best-performing one for deployment.

3.3 Model Visualization:

3.3.1 Correlation Heatmap:

- Generates a heatmap to visualize the correlation between different features in the dataset.
- Helps identify highly correlated features, which may indicate redundant information or multicollinearity.

3.3.2 Principal Coordinate Analysis (PCoA):

- Conducts PCoA to visualize the distribution of data points in a reduced-dimensional space.

- Enables visualization of complex data structures and patterns that may not be apparent in the original feature space.

3.3.3 Learning Curve Analysis:

- Constructs learning curves by varying the size of the training set and observing the model's performance.

- Helps assess the model's ability to generalize to unseen data and identify underfitting or overfitting issues.

3.4. Model Export:

Trained Model Export:

- Utilizes the `pickle` library to serialize the trained Random Forest model into a file (`model.pkl`).

- Enables easy deployment and reuse of the model for making predictions in production environments without needing to retrain it.

4. Creation of API and Deployment

4.1. Flask Web Application Architecture:

-The Flask web application serves as the backend for the thyroid prediction system. It provides a user-friendly interface for users to input relevant data and receive predictions regarding thyroid conditions.

Components:

1. Flask Application - Handles incoming HTTP requests, routes them to appropriate endpoints, and generates HTTP responses.

2. HTML Templates - Defines the structure and content of web pages served by the Flask application. These templates are rendered dynamically based on user requests.

3. Prediction Endpoint - Defines a route in the Flask application to accept POST requests containing input data for thyroid prediction.

4.2. Model Prediction Logic:

The model prediction logic handles the processing of input data and utilizes a pre-trained machine learning model to predict thyroid conditions based on the provided parameters.

Components:

1. Model Loading - Loads the pre-trained machine learning model from a serialized file (`model.pkl`) using the pickle library.

2. Input Processing - Extracts input data submitted through the web form, including patient details and thyroid-related parameters.

3. Prediction Logic - Utilizes the loaded model to make predictions on the input data, determining the likelihood of various thyroid conditions.

4. Result Generation - Converts the model predictions into human-readable format, providing an interpretation of the predicted thyroid condition.

4.3. User Interface (Web Page):

The user interface (UI) encompasses the HTML form displayed to users, allowing them to input relevant data for thyroid prediction, submit the form, and view the prediction results.

Components:

1. HTML Form - Presents a web form to users, prompting them to enter patient information and relevant thyroid-related parameters.

2. User Input - Represents the data entered by users into the HTML form, including age, medication status, laboratory measurements, and patient ID.

3. Submit Button - Initiates the submission of user input to the Flask application for thyroid prediction upon clicking.

4. Prediction Result - Displays the predicted thyroid condition generated by the Flask application in response to the user's input.

4.4. Integration and Deployment:

Integration and deployment involve hosting the Flask web application on a web server and ensuring the availability of the trained model for prediction.

Components:

1. Web Server - Hosts the Flask application, allowing users to access the thyroid prediction functionality via web browsers.

2. Model Persistence - Ensures the availability of the trained model for prediction by persisting it in a serialized format (`model.pkl`).

3. Communication Protocol (HTTP) - Defines the protocol used for communication between the web browser and the Flask application, typically HTTP or HTTPS.

4.5. Version Control and Deployment:

Version control using GitHub and deployment on Azure facilitate code management, collaboration, and application hosting.

Components:

1. GitHub Repository - Hosts the source code of the Flask application, HTML templates, and any supporting files, allowing for version control, collaboration, and code sharing.
2. Code Synchronization - Enables developers to synchronize code changes, manage branches, and track project history using Git commands or GitHub's user interface.
3. Continuous Deployment (Azure) - Utilizes Azure services for continuous deployment, automating the process of deploying updates to the Flask application whenever changes are pushed to the GitHub repository.

4. Azure Web Services - Hosts the deployed Flask application, ensuring its availability to users over the internet. Azure provides scalable infrastructure and management tools for application hosting.

By leveraging GitHub for version control and Azure for deployment, the development workflow becomes streamlined, allowing for efficient collaboration, version tracking, and seamless application deployment and management.