Low Level Design

Thyroid Detection

Written By	Saurabh Tandon
Document Version	1.0
Last Revised Date	

Document Version Control

Change Record:

Version	Date	Author	Comments
1.0	11-04-2023	Saurabh Tandon	Introduction & Architecture defined

Reviews:

Version	Date	Reviewer	Comments

Approval Status:

Version	Review Date	Reviewed By	Approved By	Comments

Contents

1. Introduction

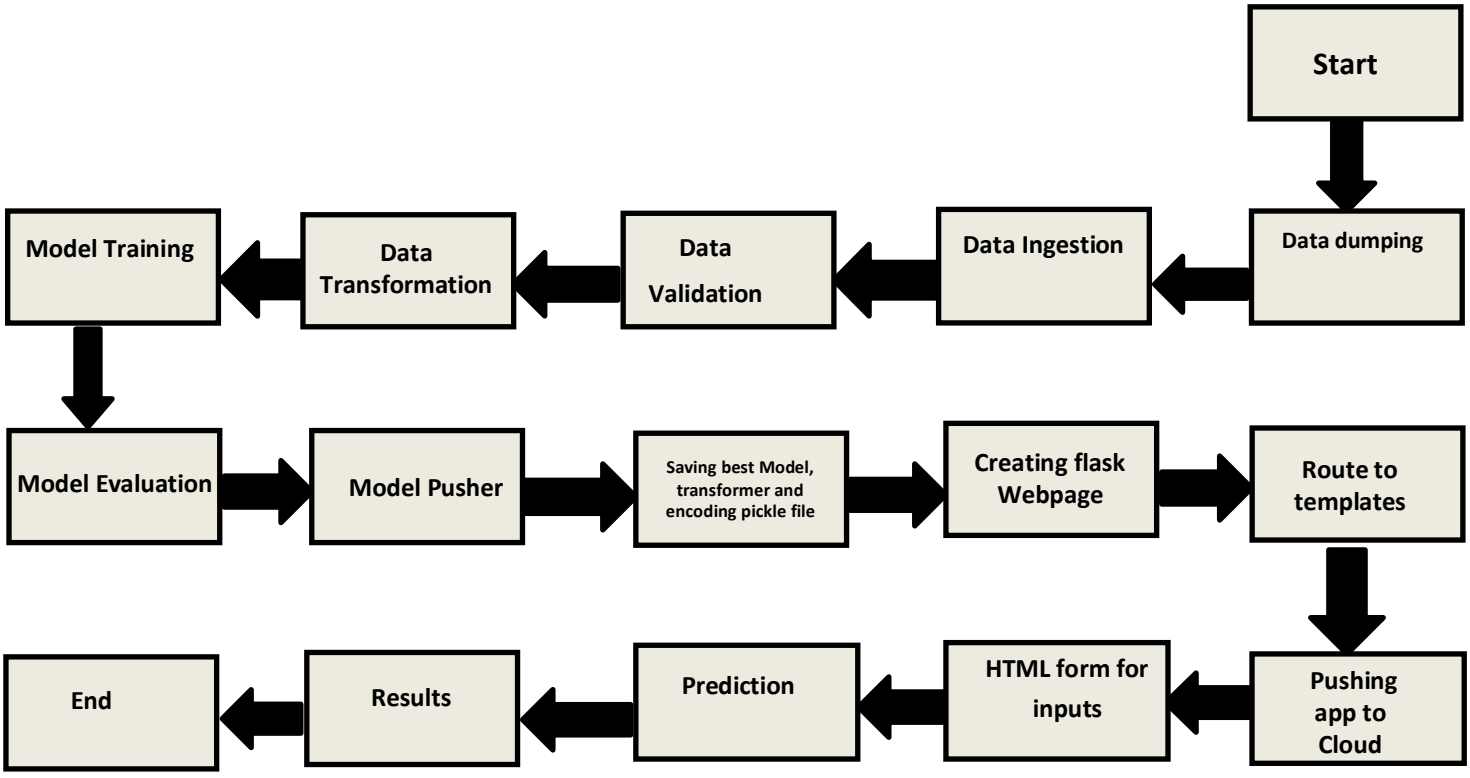
1.1 What is a Low-Level design document?

The goal of LLD or a low-level design document (LLD) is to give the internal logical design of the actual program code for Thyroid Disease Detection 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 Description

We will be using Thyroid Disease Data Set present in UCI Machine Learning Repository. This Data set is satisfying our data requirement. Total 9069 records present.

3.2 Export Data from database to CSV for Training

Here we will be exporting all batches of data from the database into one csv filefor training.

3.3 Data Validation

Here I used different cleaning of data and setting the dataset cleaned for further operations. Here I also did the splitting of data in train and test file and send it to artifacts folder.

3.4 Data Transformation

In this section I have done the feature selection and transformation of the columns such as Ordinal Encoder, One Hot Encoding, KNNImputer and Min Max Scaler and imputed on splitted data and created the object and sent it to artifacts folder.

3.5 Model Trainer

Decision Tree Classifier is selected as it gave the best accuracy after fine tuning the Hyperparameter and giving best accuracy for max_depth=8. Also creating the object and saving it to the artifacts folder.

3.6 Model Evaluation

Here I evaluated the model accuracy and also checked if the model has already had any previous object that is giving better accuracy in compared to the current object. The best model is to be saved in the artifacts folder.

3.7 Model Pusher

Here the best accurate model object is to be sent to the saved_model folder with the naming convention as the latest file object has highest numeric.

3.8 Flask Webpage

Here I created a flask webpage to take the inputs from the user and show the result with the help of the predictions. Here two HTML templates were designed.

3.9 Creating Docker File

Here the Docker image of the file created is made and the code to which file to run is written accordingly.

3.10 Using GitHub actions to create CI/CD pipeline

Here code is witten to interact with the AWS to build, tag and push Docker image to ECR and connect GitHub to AWS EC2 to run continuous deployment.

4 Unit Test Cases

Test Case Description	Pre-Requisite	Expected Result
Verify whether the Application URL is accessible to the user	1. Application URL should be defined	Application URL accessible to the user
Verify whether the Application loads completely for the user when the URL is accessed	1.Application URL is accessible 2. Application is deployed	The Application should load completely for the user when the URL is accessed
Verify whether user can see inputfields	Application is accessible.	User should be able to see input fields
Verify whether user can edit all input fields	Application is accessible.	User should be able to edit all input fields
Verify whether user gets Submit button to submit the inputs	1. Application is accessible. 2.Users are signed upto the application. 3. User is logged into the application	User should get Submit button to submit the inputs