Low Level Design

Thyroid Disease Detection

Document Control

Change Record

|  |  |  |  |
| --- | --- | --- | --- |
| Date Issue | Version | Description | Author |
| 08/01/2024 | 1 | Initial LLD – V 1.0 | Jesima Parvin M |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

Approval Status

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Version | Review Date | Review by | Approved by | Comments |
|  |  |  |  |  |

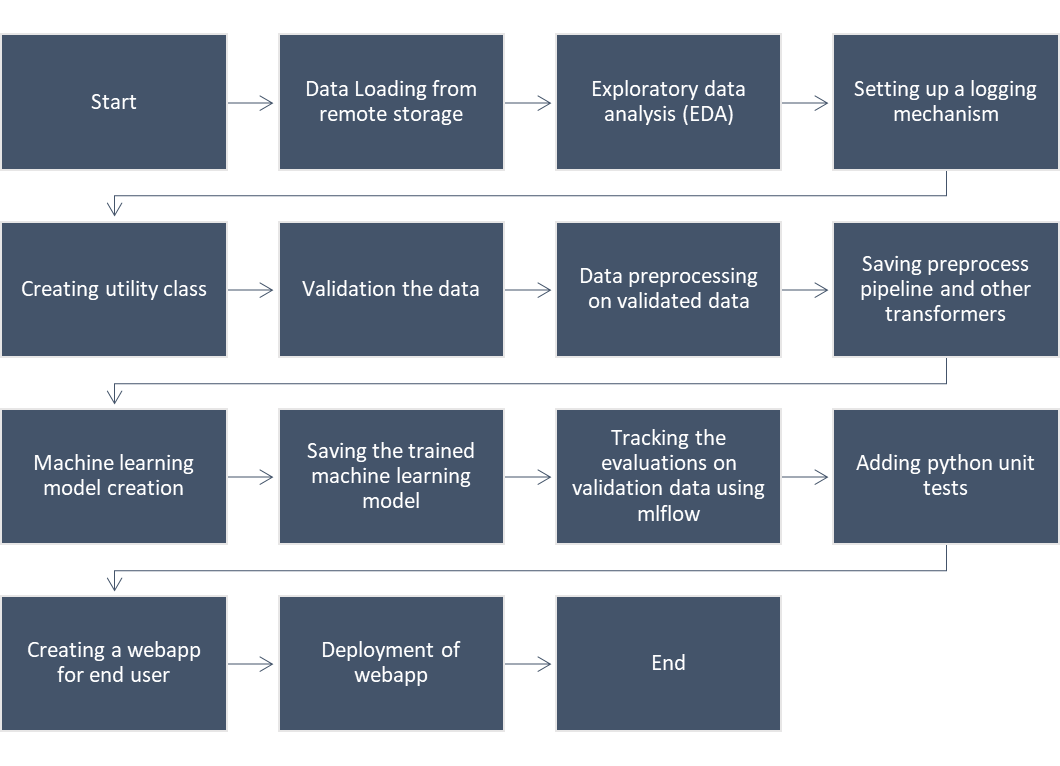
Content

1. Introduction ………………………………………………………………………………………………………………
   1. What is Low Level Design Document …………………………………………………………………..
   2. Scope ………………………………………………………………………………………………………………….
2. Architecture ……………………………………………………………………………………………………………..
3. Architecture Description …………………………………………………………………………………………..
   1. Data Description …………………………………………………………………………………………………
   2. Data Validation …………………………………………………………………………………………………..
   3. Data Pre-Processing and feature selection ………………………………………………………….
   4. Testing for Classification algorithm …………………………………………………………………….
   5. Selecting model with best ROC AUC score …………………………………………………………..
   6. Model training …………………………………………………………………………………………………….
   7. Model Evaluation ………………………………………………………………………………………………..
   8. Testing of code ……………………………………………………………………………………………………
   9. Model deployment ……………………………………………………………………………………………..
4. Unit Test Cases ………………………………………………………………………………………………………...
5. Introduction
   1. What is Low Level Design Document?

The goal of LLD or Low-Level Design Document (LLDD) is to give the internal logic design of the actual program code for Thyroid Disease Detection. LLD describes the class diagram with the methods and relations between classes and program specs. It describes the modules so that programmer can directly code from the document.

* 1. Scope

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

1. Architecture
2. Architecture Description
   1. Data description

The dataset contains 31 unique columns and around 9000 rows. Dataset is collected from UCI Machine Learning Repository.

* 1. Data Validation

Input data features are validated for their data type. The feature named ‘age’ also has one more validation regarding its range. The code will show the warning if the values of age are less than 0 or greater than 100. This is because humans live as long as 100 years and also the age is a positive integer value. Python pydantic library was used for the validation of above-mentioned criteria.

* 1. Data Preprocessing and feature selection

In this process, we will perform following operations on the raw data:

1. Removing columns that does not contribute to machine learning model training
2. Converting invalid entries of age feature to null values. The values of age less than 0 or the value greater than 100 is considered as invalid for the preprocessing purpose given the normal human being live very rarely above the age of 100.
3. The people predicted to be having no anomalies regarding thyroid as represented as ‘-‘string in target column. So for the understanding purpose, we will replace ‘-‘ string with ‘Others’ string.
4. The encoding and imputation of missing values is performed on the categorical features.
5. Imputation of missing values is performed on the numerical features.
   1. Testing for Classification algorithm

By using cross-validation in EDA, we came to know that the xgboost classifier machine learning model works best with our dataset. So, we will use the same model for the model training though we might experiment with its hyperparameters a bit to increase an accuracy. Different performance measures calculated during the experimentation phase were tracked using the mlflow python library.

* 1. Selecting model with best f1 score

The model with the best f1 score was chosen.

* 1. Model training

Model will be trained on the whole dataset and saved as a pickle file.

* 1. Model Evaluation

The trained machine learning model will be evaluated on the validation data. Different metrics were calculated and stored as a json file.

* 1. Testing of code

Many unit tests were created in order to test out all the parts of code up to this point. The python pytest library was used to create these unit tests.

* 1. Deployment

The whole solution created above will be pushed to a cloud platform for user to interact with it. For this project, we will use the ‘streamlit deploy’ service for our deployment purpose.

1. Unit Test Case

|  |  |  |
| --- | --- | --- |
| Test Case Description | Pre - requisite | Expected result |
| Verify whether application URL is accessible to the user | 1.Application URL should be defined | Application URL should be accessible to the users |
| Verify whether the application loads successfully when the URL is hit | 1. Application URL is accessible  2. Application is deployed | The application loads successfully when the URL is hit |
| Verify whether user is able to see input fields | 1. Application is  accessible | User should be able to see input fields |
| Verify whether user is able to edit all input fields | 1. Application is  accessible | User should be able to edit all input fields |
| Verify whether user gets input fields to provide the data | 1. Application is  accessible | User should input fields to  submit the inputs data |
| Verify whether user is presented with recommended results on clicking  Make a Prediction button | 1. Application is  accessible | User should be presented with  recommended results on clicking  Make Prediction button |
| Checking if data is loaded as a dataframe |  | The data should be loaded as a dataframe |
| Checking the shape of input data |  | Shape of input data should be (9172, 31) |
| Checking the shape of output data |  | Shape of output data after processing should be (8251, 23), (918, 23), (8254, 1), (918, 1) |
| Checking if the preprocess pipeline is saved in desired directory |  | Preprocess pipeline should be saved in ‘Preprocessing\_pipeline’ directory |
| Checking if the label encoder is saved in desired directory |  | Label encoder should be saved in ‘Preprocessing\_pipeline’  directory |
| Checking if the process data is saved in desired directory |  | The processed data should be saved in the ‘Data/Processed\_Data’ directory |
| Checking if the directories created during intermediate preprocessing steps are removed or not |  | Intermediate directories should be removed |
| Checking if the categorical features have been encoded |  | Categorical features should be encoded |
| Checking if there are any missing values left |  | There shouldn’t be any missing values in the data left. |
| Checking if the trained model is saved in desired directory |  | Trained model should be saved in ‘Models’ directory |
| Checking if the evaluation metrics are saved in desired directory and also check if they are in valid range |  | Evaluation metrics should be saved in ‘Metrics/metrics.json’ file and also in ‘Metrics/classification\_report.csv’ file |
| Checking if the model is underfitted or overfitted |  | We are considering the model to be underfitted if the roc auc score of train data on the trained model is less than or equal to 0.5.  We are considering the model to be overfitted if the difference between the roc auc score of train and test data is more than 0.25. |