Thyroid Disease Detection

Architecture

Written By	Ankit Singh
Document Version	1.0
Last Revised Date	5-June-2024

Contents

1. Introduction

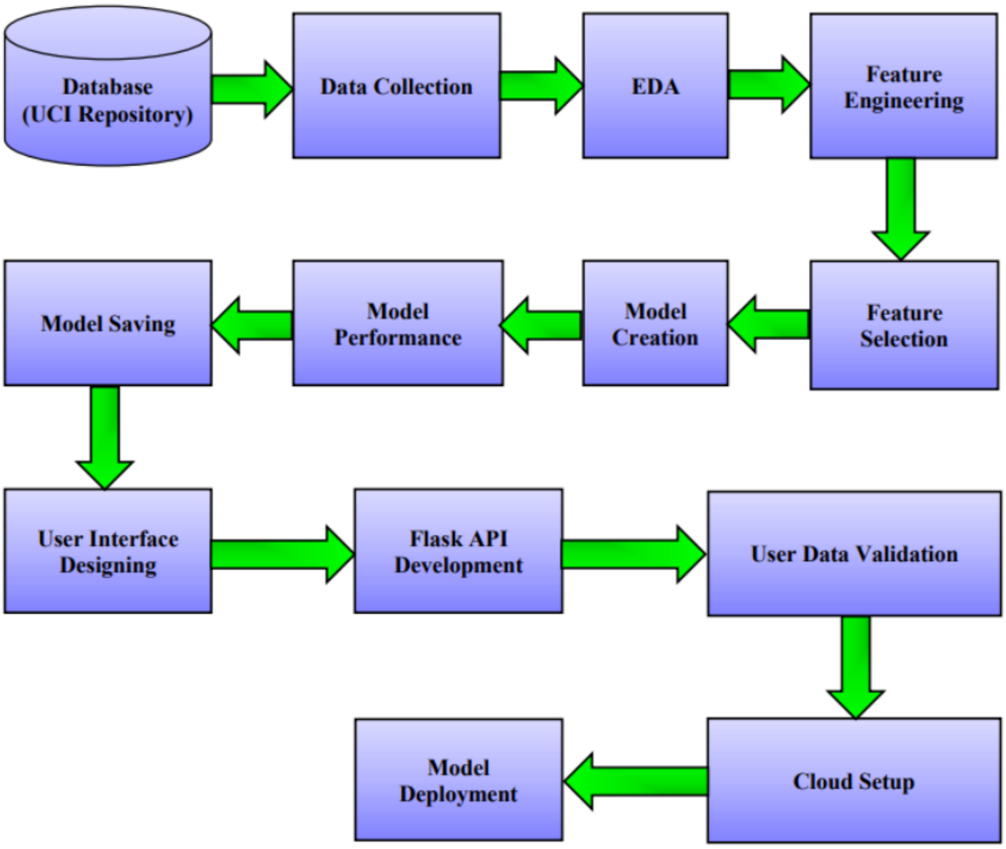
1.1 What is Low level design document?

Giving the internal logical design of the actual programme code for the Thyroid Disease Detection System is the aim of a low-level design document (LLD). LLD explains class diagrams with methods, connections between classes, and programme specifications. The modules are described in detail so that the programmer can create the software directly from the document.

1.2 Scope

Low-level design (LLD) is an iterative refinement method that focuses on component-level design. Data structures, necessary software architecture, source code, and finally performance algorithms can all be designed using this method. Overall, the data structure may be established during the analysis of the requirements and then improved upon during the process of data design.

2. Architecture



3. Architecture Description

3.1 Database (UCI Repository)

We are using Thyroid Disease Data Sets present in UCI Machine Learning Repository which is **allhypo** dataset given in csv file format. Allhypo dataset is to classify the records for hypothyroid, primary hypothyroid, compensated hypothyroid, secondary hypothyroid and negative classes.

Below is the link to access the dataset present in UCI repository:

https://archive.ics.uci.edu/ml/machine-learning-databases/thyroid-disease/

There are 2800 instances and 29 attributes in the

dataset.

Columns present in dataset are: age, sex, on thyroxine, query on thyroxine, on antithyroid medication, sick, pregnant, thyroid surgery, I 131 treatment, query hypothyroid, query hyperthyroid, lithium, goitre, tumour, hypopituitary, psych, TSH measured, TSH, T3 measured, T3, TT4 measured, TT4, T4U measured, T4U, FTI measured, FTI, TBG measured, TBG, referral source, labels.

3.2 Data Collection

I imported the csv file in Jupyter notebook using Pandas library.

3.3 EDA

In the cleaning phase, I removed all the data that wasn't already in a machine learning algorithm recognizable format. Additionally, aspects that are not relevant were removed. The Null values were processed and replaced with the mean and median value accordingly. Various libraries, including seaborn, Plotly, matplotlib, and others, were used to visualise the data.

3.4 Feature Engineering

Deleted a row which has invalid age values. Also performed One hot and label encoding to convert the categorical features to numerical features. We transformed the data using different transformations like ‘log’ and ‘square root’ transformations to remove the skewness of the data present in different features.

3.5 Feature Selection

Dropped the columns which were not required after transforming. Feature importance of each column is checked using Extra Trees classifier. And 9 columns were selected as important features that significantly affect accuracy. Features like 'age', 'sex', 'on_thyroxine', 'sick', 'query_hypothyroid', 'psych', 'T3', 'TT4', and 'FTI' were chosen for final model.

3.6 Model Creation

After completing the feature selection process, data was split between train and test data. Dataset was imbalanced having high no. of negative classes in comparison to other classes therefore used Random Over sampling technique to handle it. Different machine learning classification algorithms like random forest, k-Nearest Neighbors, Decision Tree, and XG Boost were applied on the training data and testing data.

3.7 Model Performance

After fitting the data on different models, we checked the performance using different metrics like accuracy score, precision, recall. We got roc score of 0.988 for XGBoost.

3.8 Model Saving

After comparing all accuracies, XGBoost model was considered as best model and so I dumped this model in a pickle file format.

3.9 User Interface Designing

I created a user interactive page where user can enter their input values to different fields of my application. In this front-end page was designed as form which take values using text-boxes. These HTML user input data is transferred in json format to backend. After filling the fields values, user can see the result by clicking on the predict button.

3.10 Test Case Validation

Data validation is done here which was entered by user.

3.11 Cloud Setup

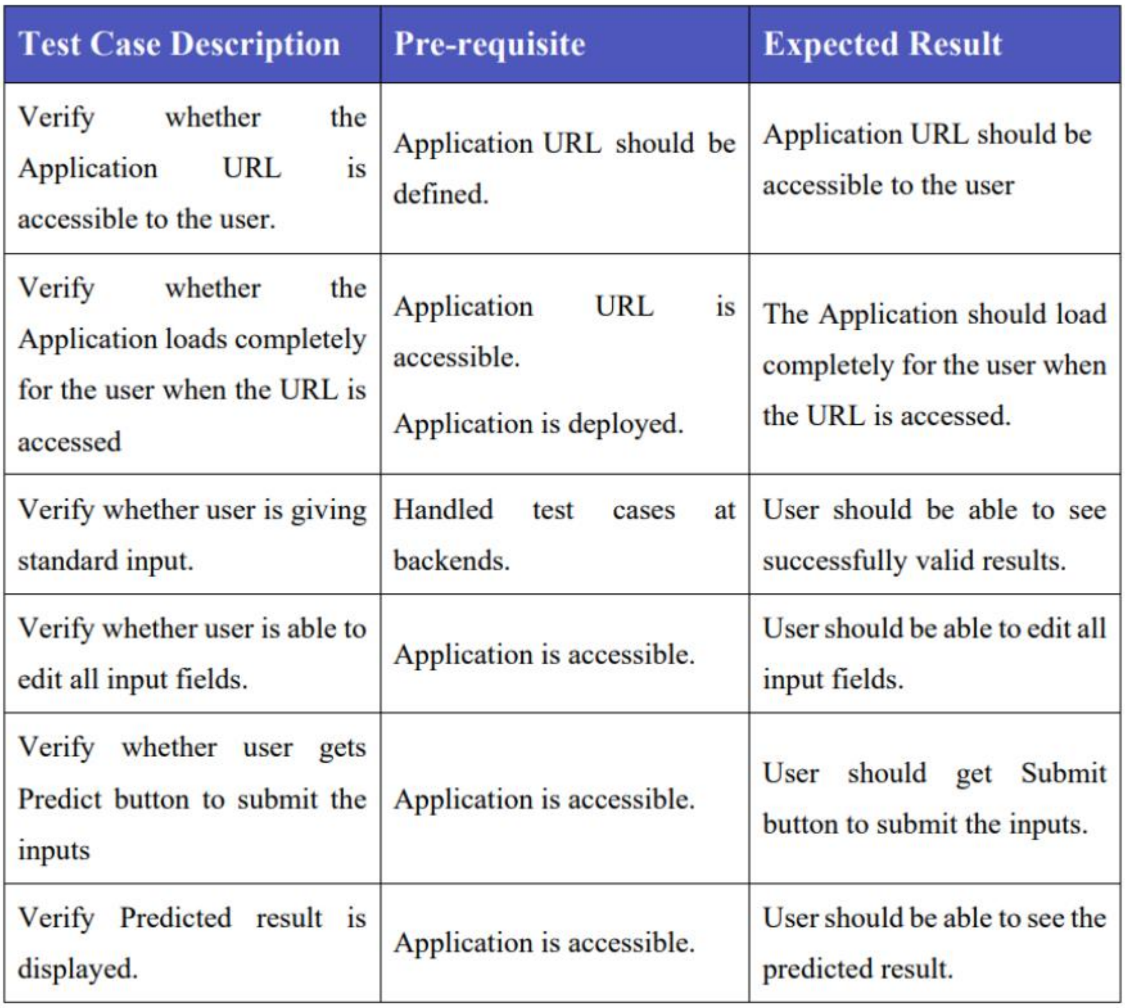
I used Microsoft Azure cloud to deploy my application.

3.12 Model deployment

Deployed my model on Azure which generate one URL. Using that URL, user can access the User Interface of the application.

URL: https://thyroiddetect.azurewebsites.net/

4. Unit Test Cases



Test Case Description	Pre-requisite	Expected Result
Verify whether the Application URL is accessible to the user.	Application URL should be defined.	Application URL should be accessible to the user
Verify whether the Application loads completely for the user when the URL is accessed	Application URL is accessible. Application is deployed.	The Application should load completely for the user when the URL is accessed.
Verify whether user is giving standard input.	Handled test cases at backends.	User should be able to see successfully valid results.
Verify whether user is able to edit all input fields.	Application is accessible.	User should be able to edit all input fields.
Verify whether user gets Predict button to submit the inputs	Application is accessible.	User should get Submit button to submit the inputs.
Verify Predicted result is displayed.	Application is accessible.	User should be able to see the predicted result.