Thyroid Disease Detection

Low Level Document (LLD)

Written By	**Ahmad Taquee, Priyanka Gupta, Diksha Sharma**
Document Version	**1.1**
Last Revised Date	**10-January-2022**

Document Control

Change Records:

Version	Date	Author	Comments
1.0	10-Jan-2021	Ahmad Taquee, Priyanka Gupta, Diksha Sharma	Introduction & Architecture defined
1.1	10-Jan-2022	Ahmad Taquee, Priyanka Gupta, Diksha Sharma	User Interface defined

Reviews:

Version	Date	Reviewer	Comments

Approval Status:

Version	Review Date	Reviewed By	Approved By	Comments

Contents

1. Introduction

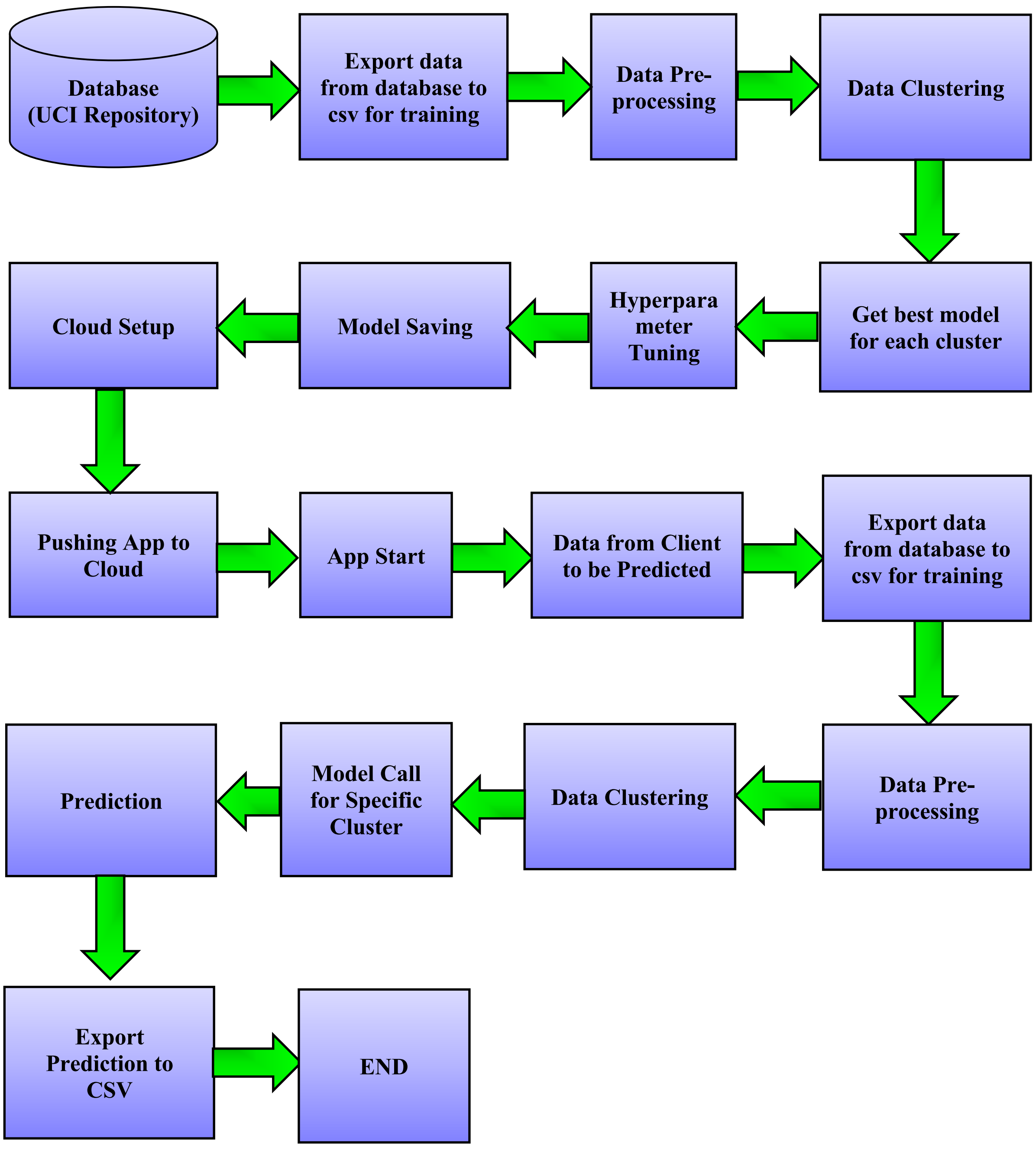
1.1. What is Low-Level design document?

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 describe 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


Database (UCI Repository)
Export data from database to csv for training
Data Pre-processing
Data Clustering
Get best model for each cluster
Hyperparameter Tuning
Model Saving
Cloud Setup
Pushing App to Cloud
App Start
Data from Client to be Predicted
Export data from database to csv for training
Data Pre-processing
Data Clustering
Model Call for Specific Cluster
Prediction
Export Prediction to CSV
END

3. Architecture Description

3.1 Database (UCI Repository) Description

We will be using Thyroid Disease Data Set present in UCI Machine Learning Repository. This Data set is satisfying our data requirement. Total 7200 instances present in different batches of data. Data is consists of categorical and numerical attributes.

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

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

There are 2800 instances and 29 attributes in both the datasets.

Columns (same in both) present in both datasets 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 Export Data from database to CSV for Training

All those batch file which after passing through data validation step get stored in Database. And here we will be exporting all those CSV files in a single CSV file for training purpose. For this we need to write a separate module.

3.3 Data Pre-processing

We will be exploring our data set here and do EDA if required and perform data pre-processing depending on the data set. We first explore our data set in Jupyter Notebook and decide what pre-processing and Validation we have to do such as imputation of null values, dropping some column, etc and then we have to write separate modules according to our analysis, so that we can implement that for training as well as prediction data.

3.4 Data Clustering

K-Means algorithm will be used to create clusters in the pre-processed data. The optimum number of clusters is selected by plotting the elbow plot. The idea behind clustering is to implement different algorithms to train data in different clusters. The K-means model is trained over pre-processed data and the model is saved for further use in prediction.

3.5 Get best model of each cluster

Here we will train various model on each cluster which we will obtain in Data Clustering step, and then will try to get best model for each cluster.

3.6 Hyperparameter Tuning

After selecting best model for each cluster, we will do hyperparameter tuning for each selected model, and try to increase performance of the models. And finally save model for each cluster.

3.7 Model Saving

After performing hyperparameter tuning for models, we will save our models as per cluster number. so that we can use them for prediction purpose and according to cluster number we will call model for making prediction over client data.

3.8 Cloud Setup

Here We will do cloud setup for model deployment. Creating all required files needed for cloud deployment of app. Here we also create our flask app and user interface and integrate our model with flask app and UI.

3.9 Push App to Cloud

After doing cloud setup and checking app locally, we will push our app to cloud to start the application. We will be deploying the model to Heroku and AWS.

3.10 Data from client side for prediction purpose

Now our application on cloud is ready for doing prediction. The prediction data which we receive from client side will be exported from DB and further will do same data cleansing process as we have done for training data using modules we will write for training data. Client data will also go along the same process of Exporting data from DB, Data pre-processing, Data clustering and according to each cluster number we will use our saved model for prediction on that particular cluster.

3.11 Export Prediction to CSV

Finally when we get all the prediction for client data, then our final task is to export prediction to csv file and hand over it to client.

3.12 Model deployment

We deployed our model to Heroku which generate one URL. Using that URL, user can access the User Interface of the application.

3.13 User Interface

a. For single user input prediction, We will make a separate UI which will take all inputs from a single user and give back the prediction there only. So that any single user can also use this system for single prediction.

b. For batch or bulk prediction, We will make a separate UI which will take a CSV file from client and make bulk prediction at a single time and save those prediction in result.csv file and UI will give option to download that particular file.

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.