THYROID DISEASE DETECTION

ARCHITECTURE

Revision Number: 1.0

Last date of revision:01-Sept-2023

Document Version Control

Date Issued	Version	Description	Author
01-Sept-2023	1.0	CompleteLLD	Sahdev Saini

Contents

1. Introduction

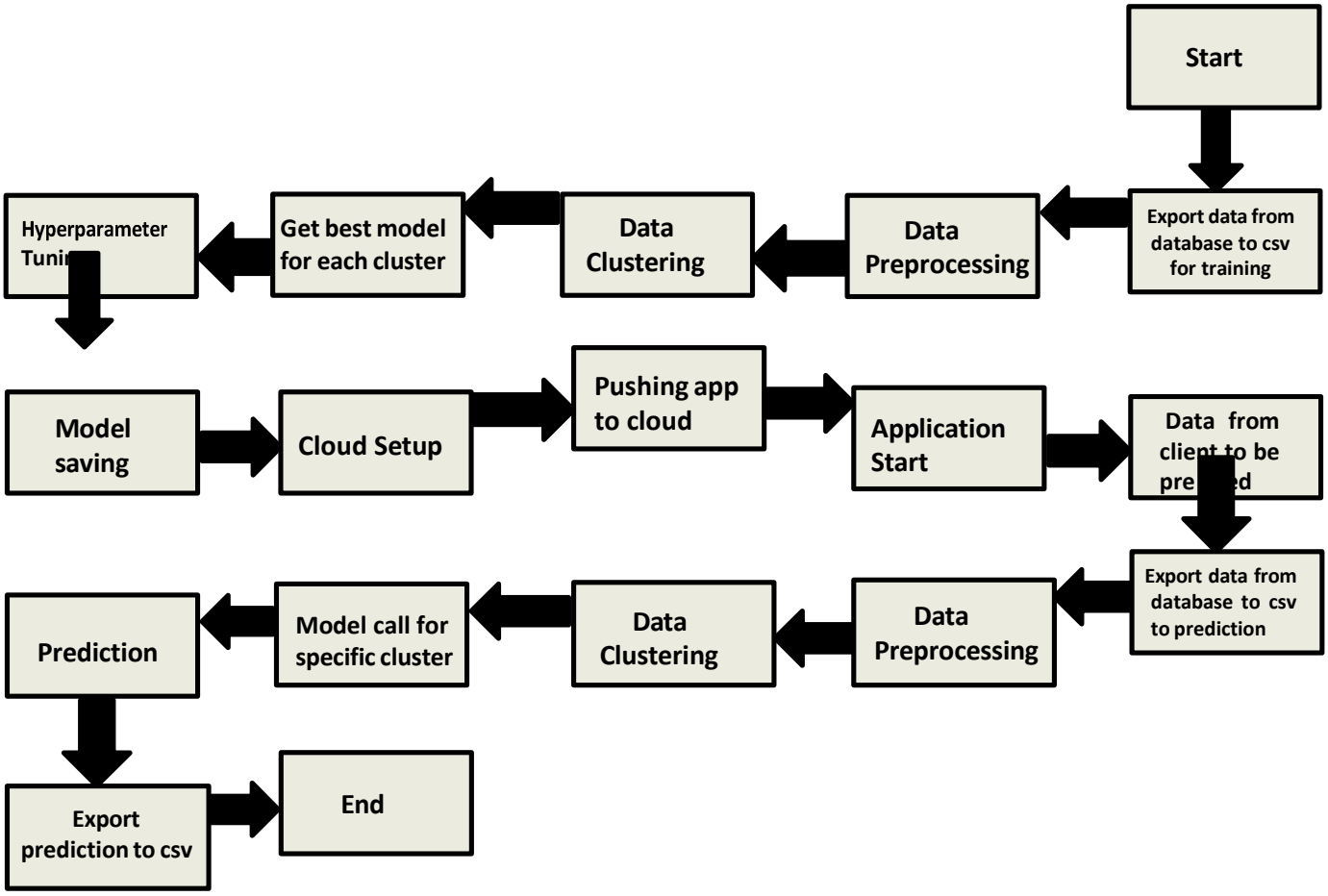
1.1 What is Low-Level design document?

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 describe 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 7200 instances present in different batches of data.

3.2 Export Data from database to CSV for Training

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

3.3 Data Preprocessing

We will be exploring our data set here and do EDA if required and perform data preprocessing 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

Random Forest is an ensemble learning method that constructs a multitude of decision trees during training. Each tree is built independently using a random subset of the features and a random subset of the training data. The final prediction is typically determined by averaging or voting among the predictions of all the individual trees. Random Forest does not involve gradient boosting; instead, it relies on bagging and random feature selection to improve predictive performance and reduce overfitting.

3.5 Get best model of each cluster

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

3.7 Model Saving

After performing hyperparameter tuning for models, we will save our models so that we can use them for prediction purpose.

3.8 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 cluster.

3.9 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