LOW LEVEL DESIGN

Restaurant Rating Prediction

Written By	Kurama Teja
Document Version	1.1

Document Control

Change Record:

version	Date	Author	Comments
1.0	23-11-2021	Kurama Teja	Introduction Architecture Architecture Description Unit Cases

Reviews:

version	Date	Reviewer	comments

Approval Status:

Version	Review date	Reviewed by	Approved by	comments

Contents

1 *Introduction*

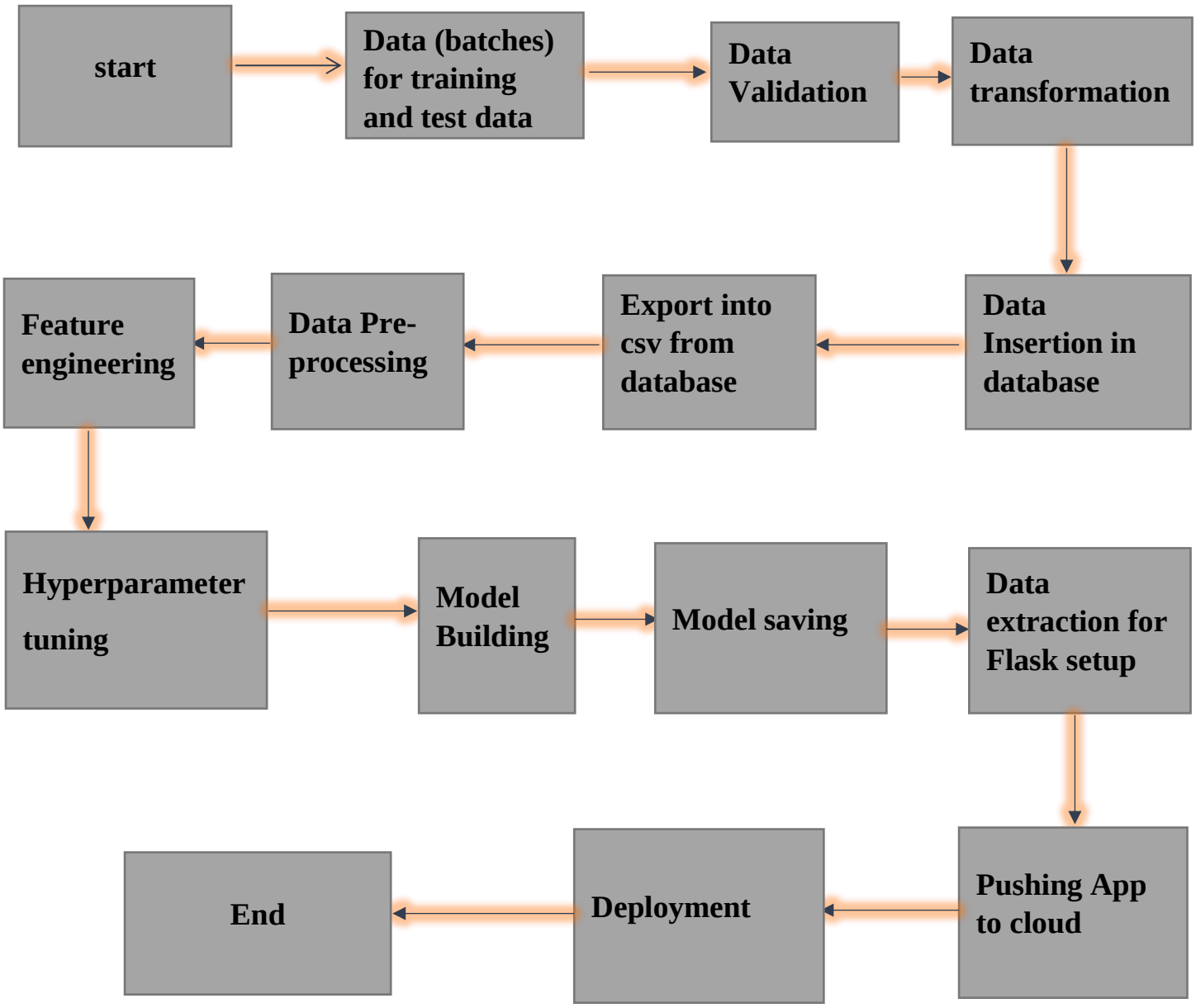
1.1 *what is low-level design document?*

The goal of LLD or a low-level design document (LLDD) is to give the internal logical design of the actual program code. Low-level design is created based on the high-level design. The code can then be developed directly from the low-level design document with minimal debugging and testing. Other advantages include lower cost and easier maintenance.

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.

2. *Architecture*



3. Architecture Description

3.1 *Data Description*

Given is the variable name, variable type, the measurement unit, and a brief description. The Restaurants' rating prediction is the regression problem. The order of this listing corresponds to the order of numerals along the rows of the database.

Name	Data Type	Measurement
URL	String	URL of the restaurant
Address	String	Address of the restaurants
Name	String	Name of the restaurants
online_order	String	Online order option is there or not in restaurants.
book_table	String	Table Booking option is there or not in restaurants.
Rate	Float	Rating of the restaurants.
Votes	String	Total number of votes restaurants have.
Phone	Integer	Phone number of restaurants.
Location	String	Location of restaurants.
rest_type	String	This column shows which type of dinning services restaurants provide.
dish_liked	String	This column shows most liked dishes of the restaurants.
Cuisines	String	This column shows which type of continental dinning services restaurants provide.
approx_cost (for two people)	Float	This column shows approximate cost for two people.
reviews_list	String	Review of the restaurants.
menu_tem	String	List of menu items of restaurants.
listed in(type)	String	Type of dinning style is listed in this column.
listed in(city)	String	City in which restaurant is present is listed here.

3.2 *Data Gathering*

Data source: https://www.kaggle.com/himanshupoddar/zomato-bangalore-restaurants Train and Test data are stored in .csv format.

3.3 *Raw Data Validation*

After data is loaded, various types of validation are required before we proceed further with any operation. Validations like checking for zero standard deviation for all the columns, checking for complete missing values in any columns, etc. These are required because The attributes which contain these are of no use. It will not play role in contributing to the sales of an item from respective outlets.

Like if any attribute is having zero standard deviation, it means that's all the values are the same, its mean is zero. This indicates that either the sale is increasing or decrease that attribute will remain the same. Similarly, if any attribute is having full missing values, then there is no use in taking that attribute into an account for operation. It's unnecessary increasing the chances of dimensionality curse.

3.4 Data Transformation

Before sending the data into the database, data transformation is required so that data are converted into such form with which it can easily insert into the database. Here, the 'Item Weight' and "Outlet Type' attributes contain the missing values. So, they are filled in both the train set as well as the test set with supported appropriate data types.

3.5 *Database Insertion*

Both train and test data set are inserted into the database. Here MongoDB database is used to store the data set. Separate collections were created for both train and test sets.

3.6 Export as 'csv' From Database

From the database both the train and test data set are exported into the local system and stored into CSV files. Now this CSV file will have proceeded for further processing.

3.7 *Data Preprocessing*

In data preprocessing all the processes required before sending the data for model building are performed. Like, here the rate column was having rating as eg:4.5/5, therefore we have removed /5 from the rating of the rate column. After that there were missing values in rate, cost, location and rest_type column which we have dropped.

3.8 *Feature Engineering*

After preprocessing it was found that some of the attributes are not important to the item sales for the particular outlet. So those attributes are removed. Even level encoding is also performed to convert the categorical features into numerical features.

3.9 Hyperparameter Tuning

Parameters are tuned using Randomized searchCV. Two algorithms are used in this problem, Random Forest, and Extra tree regressor. The parameters of all these 2 algorithms are tuned and passed into the model.

3.10 Model Building

After doing all kinds of preprocessing operations mention above and performing scaling and hyperparameter tuning, the data set is passed into all two models, Random Forest, and Extra tree regressor. It was found that Extra tree regressor performs best with the smallest RMSE value i.e. and the highest R2 score equals 0.93. So 'Extra tree regressor' performed well in this problem.

3.11 Model Saving

Model is saved using pickle library in `. pkl` format.

3.12 Data Extraction for Flask Setup

After saving the model, the API building process started using Flask. Web application creation was created here. Whatever the data user will enter and then that data will be extracted by the model to predict the prediction of sales, this is performed in this stage.

3.13 *Pushing the App to Cloud*

Finally, the whole content will be sent to cloud platform and then project directory will be pushed into GitHub repository.

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 should be 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 a user is able to see input fields while opening the application	1. Application is accessible 2. The user is able to see the input fields	Users should be able to see input fields on logging in
Verify whether a user is able to enter the input values.	1. Application is accessible 2. The user is able to see the input fields	The user should be able to fill the input field
Verify whether a user gets predict button to submit the inputs	1. Application is accessible 2. The user is able to see the input fields	Users should get Submit button to submit the inputs
Verify whether a user is presented with recommended results on clicking submit	1. Application is accessible 2. The user is able to see the input fields. 3. The user is able to see the submit button	Users should be presented with recommended results on clicking submit
Verify whether a result is in accordance with the input that the user has entered	1. Application is accessible 2. The user is able to see the input fields. 3. The user is able to see the submit button	The result should be in accordance with the input that the user has entered