DATA SCIENCE: MACHINE LEARNING PROJECT

Data Science Open Internship, INeuron.AI

Low Level Design Report

On

BACKORDER PREDICTION

Made by**: Varun Sagar Theegala**

Main Technology**: Machine Learning**

Domain**: E-commerce**

Project Start Date**: 14th August 2021**

TABLE OF CONTENTS

1. INTRODUCTION

1.1 What is a Low-Level Design Document?

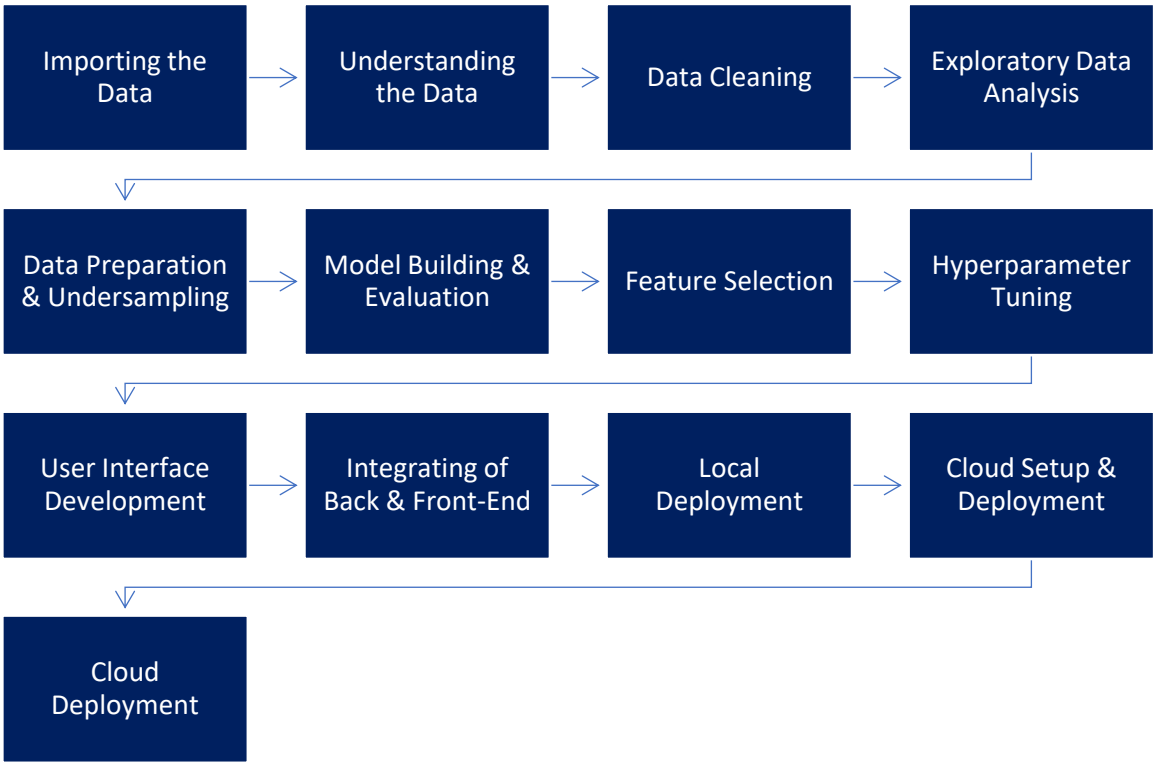
The goal of a Low-Level design (LLD) document to lay out the path following which any programmer or user can rebuild the program, including the prediction model for Backorders. An LLD describes the steps, objects created, modules imported, and tools used to create the entire program & interface. It acts as a design or blueprint for the users.

1.2 Scope of a Low Level Design Document

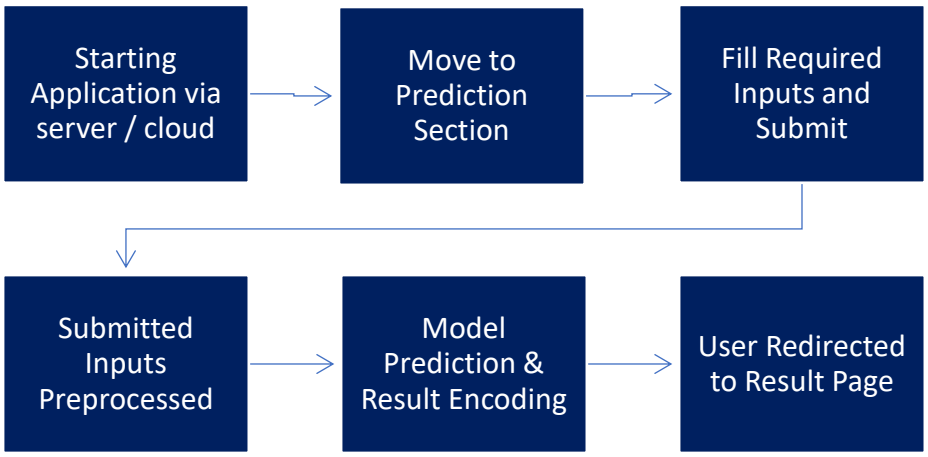
A Low-Level Design document will facilitate users to create the entire program from scratch in a systematic process. It lays out all required components, on which the program was built, for the user to write or import. On an organization level, an LLD can be the starting point for developers to define the data, software & performance requirements. A Low-Level Design would also give a reference to users or organization for the purpose of refining the code or making changes as per their needs.

2. ARCHITECTURE

2.1 Development Process



2.2 Running Application



3. ARCHITECTURE DESCRIPTION

3.1 Development Process

3.1.1 Importing the Dataset

The dataset for this project was in the form of a csv file, which was available and taken from a GitHub Repository. The dataset was imported into Amazon Sagemaker notebook as a DataFrame using the Pandas library

3.1.2 Understanding the Dataset

The dataset was thoroughly understood in respect to the no. of records, no. of columns, the data types of columns, the no. of unique values in each feature, presence of missing values & the proportion of target classes.

3.1.3 Data Cleaning

The column names were checked for lead spaces or any special characters or symbols which could cause problems while calling the columns. Also, completely empty records were identified and removed.

3.1.4 Exploratory Data Analysis

To determine whether the independent features in the dataset were sufficient to identify backorders, a detailed exploratory data analysis was conducted. Univariate Analysis was used to understand the distribution characteristics or category proportions of the independent features. Bivariate Analysis was conducted to understand the relationship between the predicted feature and predictors. Multivariate Analysis was conducted to map the target feature onto the relationship between 2 or more independent features and check if backorders can be identified. Also, multicollinearity in the dataset was checked and accordingly treated. These tasks were performed using visualization plots, cross tabulations, aggregations, and inferential statistics.

3.1.5 Data Preparation & Under-sampling

The dataset was split into train and test sample for data preparation. The training sample was used to check missing values, outliers, need for encoding or transformation. Accordingly, appropriate methods were fit onto the training sample to learn the defects and transformations were made to training and testing samples. The samples were combined, and under-sampling of majority target class were performed to get a reduced proportion between minority and

majority class to improve our model's performance. Finally, the dataset was split into train and test samples for model building.

3.1.6 Baseline Model Building & Evaluation

Machine Learning algorithms like Logistic Regression, K-Nearest Neighbours, SVM, Decision Tree, Bagging and Boosting Ensemble techniques were used to create classifiers to differentiate between backorders and non-backorders. Performance of these models were evaluated using F1 scores due to slight presence of imbalance despite under-sampling. ROC-AUC Scores and curve were utilized to understand each model's True Positive and False Positive rates. Stability of the model was evaluated using cross validation samples. Also, response time of the model for making prediction for 1 input was measured in seconds. At last, the Random Forest Model was selected based on multiple criterions.

3.1.7 Feature Selection

To make the models less complex and more interpretable, feature selection techniques like Backward Feature Elimination, Recursive Feature Elimination and Feature Importance scores were used. Finally, the model built using top 10 features based on the Feature Importance scores was selected for lesser features, similar performance, and simpler model architecture.

3.1.8 Hyperparameter Tuning

To boost the model's performance, hyperparameter tuning was performed using the Grid Search CV method. However, performance of tuned model wasn't better than previous baseline models. The final model selected was the baseline Random Forest Classifier with top 10 features (based on Feature Importance)

3.1.9 User Interface Development

The front-end interface to interact with the model. It provides a section containing information about Backorders and its Significance for an E-commerce entity and another section consisting of a form with input fields related to a product, that a user wishes to identify as a backorder or not. HTML and CSS were used to build and style the interface whereas JavaScript was used to add a few components like navigation bar, scrolling to different sections, etc.

3.1.10 Integrating Back & Front-End

To connect the front-end interface to the model running using Python, the Flask web framework was used to create an application, with the Python script initiating the front-end interface and model, are wrapped inside.

3.1.11 Local Deployment

To make the web application accessible to users, it was deployed in 2 different environments. First, it was deployed on the internet for any internet user to use with the help of Heroku App. All the project files were added to a GitHub repository and connect to an app created on Heroku for the application.

3.1.12 Cloud Setup & Development

To host the application in cloud, an AWS Ec2 instance was initiated and with the help of other programs like PuTTYgen, PuTTY and WinSCP, the application was pushed to the cloud.

3.2 Running Application Process

3.2.1 Starting Application via server / cloud

The application can be started by using the Heroku app URL or the Public IP address for the cloud platform.

3.2.2 Moving to Prediction Section

On starting the application, the user will land on the home page. Using the navigation pane or the scroll button on the landing page, the user can move to the prediction section

3.2.3 Fill Required Inputs and Submit

A form of input fields would be visible, which the user needs to fill and submit. The fields have been designed in a way that passing input in a wrong manner wouldn't be allowed. Also, the units in which answer should be given is clearly mentioned in the question.

3.2.4 Submitted Inputs Pre-processed

Once the input fields are submitted, the numerical features will be converted into a log normal distribution using the power transformation method. Also, the categorical features will be encoded as 1 or 0 depending on filled inputs. Once the input features have been transformed, it will be passed to the model.

3.2.5 Model Prediction & Result Encoding

The model will process the input and make a prediction. However, the prediction will originally be in the format of either 0 or 1. To make interpretation of the

result easy for the user, the result will be encoded. The result 0 will be encoded as “Not a Backorder” whereas 1 will be encoded as “A Backorder”.

3.2.6 User Redirected to Result Page

Once the prediction is complete, the user will be re-directed to another webpage where the transformed result will be displayed.

4. UNIT TEST CASES

Test Case Description	Pre-Requisite	Expected Result
Access to Application URL	1. Application URL should be defined on project platform like GitHub or in LinkedIn posts related to the project.	Application URL should be accessible to the user.
Application Loading	1. Application URLs for the cloud and server should be available. 2. The Application should have been successfully deployed using Heroku. 3. The application should have been successfully hosted in cloud using Ec2.	On using the URLs, the web application should start.
Interface Performance	1. The links to the folders for CSS, templates and JavaScript should be present in the Python file. 2. The folders should be properly pushed to the cloud platform and GitHub platforms.	On the start of the application, the web pages should load properly. The styling of the pages should appear as per mentioned in CSS file. Scrolling mechanism and navigation bar should be visible.
Input Filling	1. The Application should have been started. 2. The user should have scrolled to the Prediction section	On filling the input fields in the correct form, the user should see values in the field.
Input Submission	1. All mandatory Inputs fields should be filled	On pressing the submit button, the inputs should get submitted and user should get redirected to another webpage.

Prediction Generation & Display.	1. The Inputs should get successfully submitted. 2. Inputs should be appropriately transformed based on their type (i.e numerical or categorical) 3. The model should successfully process the inputs and generate a result. 4. The results should be encoded.	The user should be redirected to a new webpage where they should see either “Not a Backorder” if model has originally predicted 0 or “A Backorder” if model has originally predicted 1.