Low Level Design

Scania Truck Failures Predictions

Document Control

Change Record

|  |  |  |  |
| --- | --- | --- | --- |
| Date Issue | Version | Description | Author |
| 20/02/2023 | 1 | Initial LLD – V 1.0 | Shivam Shinde |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

Approval Status

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Version | Review Date | Review by | Approved by | Comments |
|  |  |  |  |  |

Content

1. Introduction ………………………………………………………………………………………………………………
   1. What is Low Level Design Document …………………………………………………………………..
   2. Scope ………………………………………………………………………………………………………………….
2. Architecture ……………………………………………………………………………………………………………..
3. Architecture Description …………………………………………………………………………………………..
   1. Data Description …………………………………………………………………………………………………
   2. Data Pre-Processing and feature selection ………………………………………………………….
   3. Testing for Classification algorithm …………………………………………………………………….
   4. Selecting model with best ROC AUC score …………………………………………………………..
   5. Model training …………………………………………………………………………………………………….
   6. Automating model training using DVC…………………………………………………………………
   7. Model deployment ……………………………………………………………………………………………..
4. Unit Test Cases ………………………………………………………………………………………………………...
5. Introduction
   1. What is Low Level Design Document?

The goal of LLD or Low Level Design Document (LLDD) is to give the internal logic design of the actual program code for Predict Bank Credit Risk. LLD describes the class diagram with the methods and relations between classes and program specs. It describes the modules so that programmer can directly code from the document.

* 1. Scope

Low Level Design (LLD) is a component level design process that follows a step by step refinement process. This process can be used to design data structure, required software architecture, source code and ultimately performance algorithm. Overall, the data organization may be defined during requirement analysis and then refined during data design work.

1. Architecture

1. Architecture Description
   1. Data description

The dataset contains 171 unique columns and 60000 rows. Dataset is collected from UCI Machine Learning Repository.

* 1. Data Preprocessing and feature selection

In this process, we will perform following operations on the raw data:

1. Removing columns having more than 20% of missing values.
2. Replacing ‘na’ string the data with numpy null value i.e., numpy.nan.
3. Replacing missing values in the remaining columns with the median of respective column
4. Scaling the numerical columns (in this case all the columns are numerical).
   1. Testing for Classification algorithm

Here rather than trying to find the best model for the data, we will use the Random Forest classifier model with some regularization on the tree depth. This is because the random forest classifier is known to work very well on the imbalanced data and it is fact that our data is highly imbalanced.

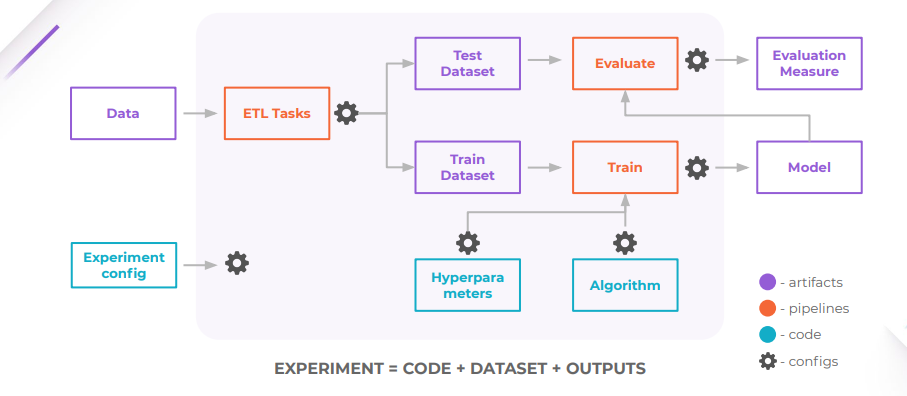
* 1. Selecting model with best ROC AUC Score

Random Forest Classifier with the highest ROC AUC score will be chosen

* 1. Model training

Model will be trained on the whole dataset and saved as a pickle file.

* 1. Automating model training and evaluation pipeline using DVC

We will use DVC python library to automate the whole model training and evaluation pipeline.

Source: DVC documentation

We will create a file named ‘params.yaml’ that will contain all the variable and hyperparameters for our code to run automatically. Then we will create a pipeline in the file named ‘dvc.yaml’. This file will contain the terminal commands to run each block of code, the dependencies, metrics, plots and outputs.

* 1. Deployment

The whole solution created above will be pushed to a cloud platform for user to interact with it. For this project, we will use the ‘streamlit deploy’ service for our deployment purpose.

1. Unit Test Case

|  |  |  |
| --- | --- | --- |
| Test Case Description | Pre - requisite | Expected result |
| Verify whether application URL is accessible to the user | 1.Application URL should be defined | Application URL should be accessible to the users |
| Verify whether the application loads successfully when the URL is hit | 1. Application URL is accessible  2. Application is deployed | The application loads successfully when the URL is hit |
| Verify whether user is able to see input fields | 1. Application is  accessible | User should be able to see input fields |
| Verify whether user is able to edit all input fields | 1. Application is  accessible | User should be able to edit all input fields |
| Verify whether user gets Submit  button to submit the inputs | 1. Application is  accessible | User should get Submit button to  submit the inputs |
| Verify whether user is presented with recommended results on clicking  submit | 1. Application is  accessible | User should be presented with  recommended results on clicking  submit |
| Verify whether the recommended  results are in accordance to the  selections user made | 1. Application is  accessible | The recommended results should  be in accordance to the selections  user made |