05/07/2023

Credit Card Default Project

Low Level Document (LLD)

Yash Kumar Singh
Vaibhav Srivastava

Contents

Abstract

Financial threats are displaying a trend about the credit risk of commercial banks as the incredible improvement in the financial industry has arisen. In this way, one of the biggest threats faced by commercial banks is the risk prediction of credit clients. The goal is to predict the probability of credit default based on credit card owner's characteristics and payment history.

Use of various classification models like Decision Tree, Random Forest, XGBoost, MLPClassifier was done. XGBoost and CatBoost were selected as best models from above. The XGBoost model provided 82.63% accuracy in training. CatBoost model provided 81% of accuracy. Testing accuracy for model was 82.3%.
70% data was used for training and 30% data was used for testing.

After Providing various details the model will predict whether the transaction is fraud or not.

1. Introduction

1.1 What is Low-Level Design Document?

The goal of LLD or a Low-level design document is to give an internal logical design of the actual program code for the Credit Card Default Probability Prediction. LLD describes the class diagrams with the methods and relations between classes and the program specs. It describes the modules so that the programmer can directly code the program from the document.

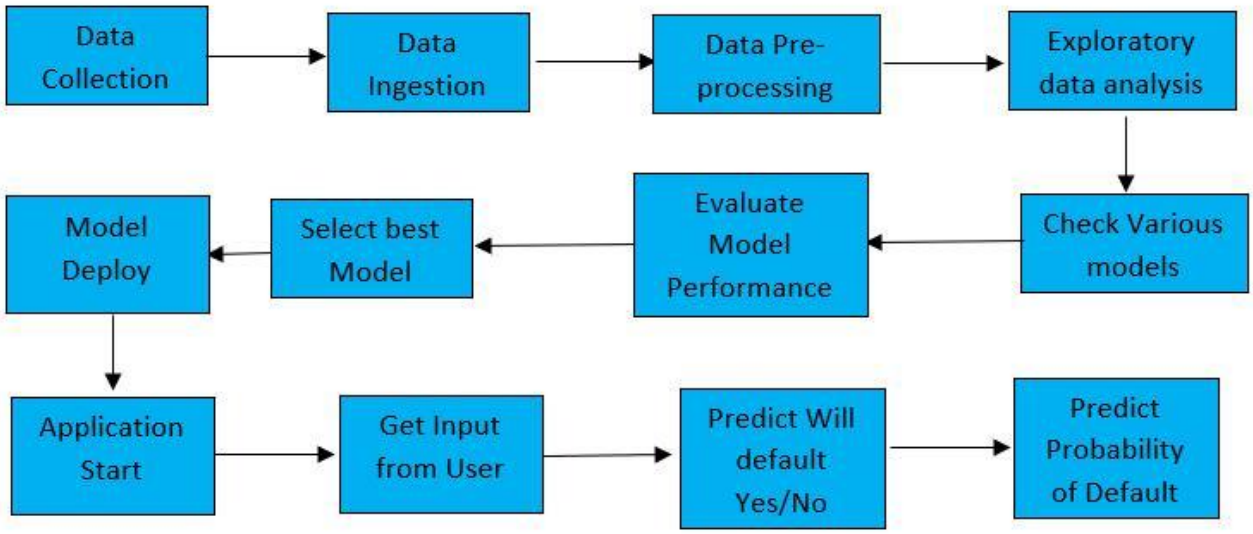
1.2 Scope

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 defined during data design work.

1.3 Definitions

Term	Description
IDE	Integrated Development Environment
EDA	Exploratory Data Analysis
XGBoost	Extreme Gradient Boost Algorithm
ML	Machine Learning
CatBoost	Categorical Boosting
VS Code	Visual Studio Code

2. Architecture



3. Architecture description

3.1 Data Collection

For training and testing the model I used a publicly available dataset on Kaggle. The dataset contains features such as User, Card, Year, Month, Day, Amount, Use Chip, Merchant Name, Merchant City, Merchant State, Zip, MCC, Errors and output feature 'is Fraud?'

Dataset Link - [Credit Card Transactions | Kaggle]

3.2 Data Sample:

User	Card	Year	Month	Day	Time	Amount	Use Chip	Merchant Name	Merchant City	Merchant State	Zip	MCC	Errors?	Is Fraud?
0	0	2002	9	1	06:21	$134.09	Swipe Transaction	3527213246127876953	La Verne	CA	91750.0	5300	NaN	No
0	0	2002	9	1	06:42	$38.48	Swipe Transaction	-727612092139916043	Monterey Park	CA	91754.0	5411	NaN	No
0	0	2002	9	2	06:22	$120.34	Swipe Transaction	-727612092139916043	Monterey Park	CA	91754.0	5411	NaN	No
0	0	2002	9	2	17:45	$128.95	Swipe Transaction	3414527459579106770	Monterey Park	CA	91754.0	5651	NaN	No
0	0	2002	9	3	06:23	$104.71	Swipe Transaction	5817218446178736267	La Verne	CA	91750.0	5912	NaN	No

3.3 Data Ingestion

For loading dataset in coding environment, we used the data_ingestion.py file that ingests the data from the desired source in required data formats.

3.4 Data Pre-Processing

Once the Data is Ingested, we transform the data in data_transformation.py file using below steps for data pre-processing that are applied in the form of a pipeline, using a preprocessor.pkl object.

Fill in the missing values: several columns in the dataset had missing values, missing values are imputed accordingly rather that dropping the rows.

Convert variables: MERCHANT STATE and ERRORS to categorical type. AMOUNT, TIME are converted to numerical datatypes.

Categorical values are encoded using Binary Encoder

All the columns are then converted to ‘float’ datatype.

Perform Label Encoding on output features.

Data Pre-processing is done.

3.5 Model Building

We have used tree based algorithms : XGBoost and Catboost.

Each of above models were built in model_trainer.py file and trained in training_pipeline.py file.

To evaluate the model, We split the data into 70% training and 30% testing using train_test_split function. Then we evaluated various metrics on training and testing data.

Multiple models such as : Logistic regression, Decision trees, SVM, naïve bayes , KNNs were tried and among them 2 most suitable algorithms were selected.

These two algorithms efficient both in time complexity as well as accuracy.

The model parameters can be checked in model_trainer.py file.

Trained models are stored in artifacts folder as model.pkl.

By default, catboost model is used due to its speed which comes at a cost of 2% macro f1 score.

3.6 Metrics Used for Model Evaluation

We used Macro f1 score and Classification report for choosing the best model.

We also used Classification report for evaluating the model.

A Classification report is used to measure the quality of predictions from a classification algorithm. How many predictions are True and how many are False. More specifically, True Positives, False Positives, True negatives, and False Negatives are used to predict the metrics of a classification report.

Macro F1 was selected as a metric as it provides a balanced evaluation between precision and recall. While we want to minimize the fraud transactions as much as possible, we do not want to impact the customer experience in the process by classifying non fraud transactions as fraud.

The performance metrics of models can be checked in log folder by the logs generated during the training process.

3.7 Saving the Best Model

The Best Model was XGBoost Classifier but it took more time in training and prediction than Catboost.

XGBoost:

	precision	recall	f1-score	support
0	1.00	1.00	1.00	7307107
1	0.72	0.61	0.66	8963
accuracy			1.00	7316070
macro avg	0.86	0.80	0.83	7316070
weighted avg	1.00	1.00	1.00	7316070

CatBoost:

	precision	recall	f1-score	support
0	1.00	1.00	1.00	7307107
1	0.73	0.53	0.61	8963
accuracy			1.00	7316070
macro avg	0.86	0.77	0.81	7316070
weighted avg	1.00	1.00	1.00	7316070

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 .	The Application should load completely for the user when the URL is accessed
Verify whether the User is able to upload the dataset to be predicted on the application	1. Application is accessible to upload the dataset. 2. Dataset is in CSV format.	The User should be able to upload the dataset.
Verify whether user is able to successfully predict the outcomes.	1. Dataset contains required features.	User should be able to successfully predict the output features.
Verify whether user is able to see input fields and output features on in log files.	1. Application is accessible 2. User has uploaded the dataset. 3. The dataset is in required format.	User should be able to see input fields and output features in log files.
Verify whether user is able to get the predicted dataset	1. Application is accessible 3. User has uploaded the dataset. 3.The dataset is in required format.	User should be able to get the predicted dataset.