CREDIT CARD DEFAULTER PREDEICTION

ARCHITECTURE

AUTHOR

RAHUL YADAV

Contents

Abstract

The financial industry has witnessed significant advancements, leading to the emergence of various financial threats, one of which is the credit risk faced by commercial banks. Accurate prediction of credit default risk has become crucial for these institutions to mitigate potential losses. This study aims to predict the probability of credit default based on the characteristics and payment history of credit card owners. By leveraging data-driven techniques, the research seeks to develop a predictive model that can accurately assess the creditworthiness of clients, thereby enabling banks to make informed lending decisions and manage their credit risk more effectively.

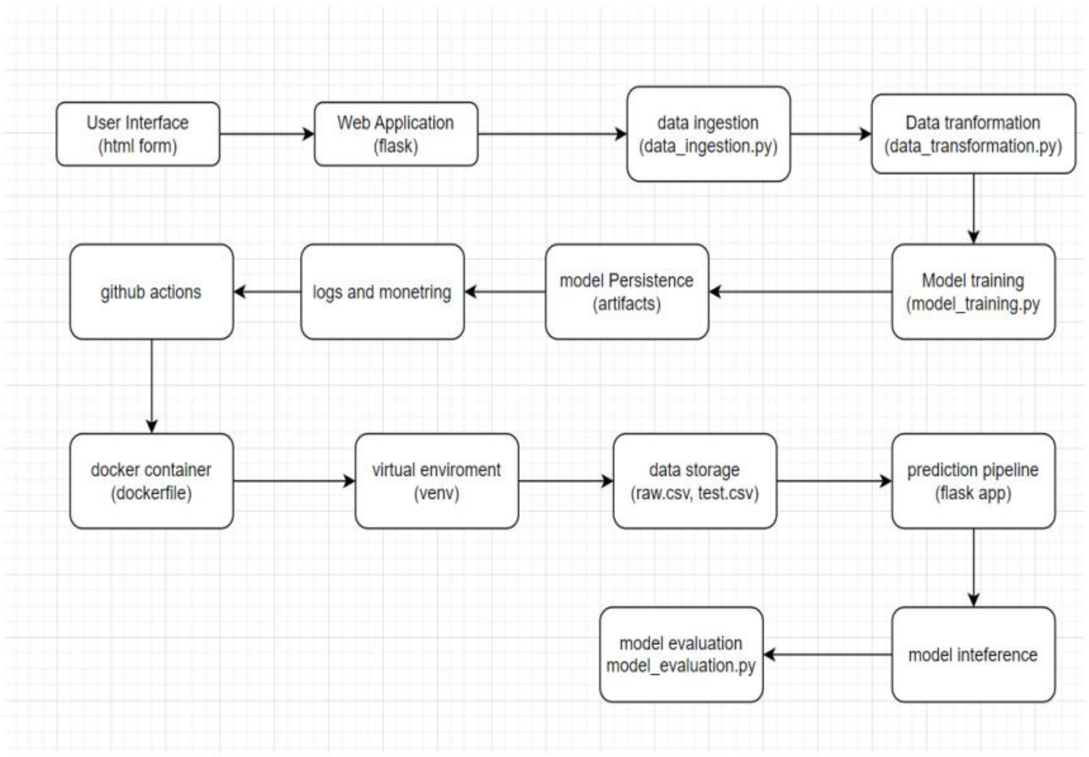
Introduction

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 for Food Recommendation System. LLD describes the class diagrams with the methods and relations between classes and 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 refined during data design work

2. Architecture


User Interface (html form)
Web Application (flask)
data ingestion (data_ingestion.py)
Data tranformation (data_transformation.py)
Model training (model_training.py
model Persistence (artifacts)
logs and monetring
github actions
docker container (dockerfile)
virtual enviroment (venv)
data storage (raw.csv, test.csv)
prediction pipeline (flask app)
model inteference
model evaluation model_evaluation.py

3. Architecture Description

3.1 Data Ingestion
The raw data is ingested using data_ingestion.py.

3.2 Data Transformation:
The ingested data is transformed and preprocessed using data_transformation.py.

3.3 Model Training:
The preprocessed data is used to train the model using model_trainer.py and the process is documented in model_training.ipynb.

3.4 Model Evaluation:
The trained model is evaluated using model_evaluation.py.

3.5 Prediction Pipeline:
The prediction_pipeline.py handles the predictions using the trained model.

3.6 Web Application: The Flask web application is defined in app.py and uses HTML templates from the templates directory to provide a user interface for input and displaying results.

3.7 CI/CD:
The .github-workflow-main.yaml file defines the CI/CD pipeline using GitHub Actions.

3.8 Version Control:
Data and models are versioned using DVC.

3.9 Logging and Monitoring:

Logs are stored in the logs directory, and experiments are tracked using MLflow in the mlruns directory.

3.10 Deployment

This project is deployed on an AWS EC2 server, allowing it to serve predictions and interact with users via a web application interface. Below are the key components and steps involved in deploying this machine learning application on an EC2 instance.

4. TECHNOLOGY STACK

FRONTEND	HTML
BACKEND	PYTHON/FLASK
DATABASE	MONGODB/CSV
DEPLOYMENT	AWS