MLOps Project

ML Deployment for Bank Churn Analysis

1. Introduction

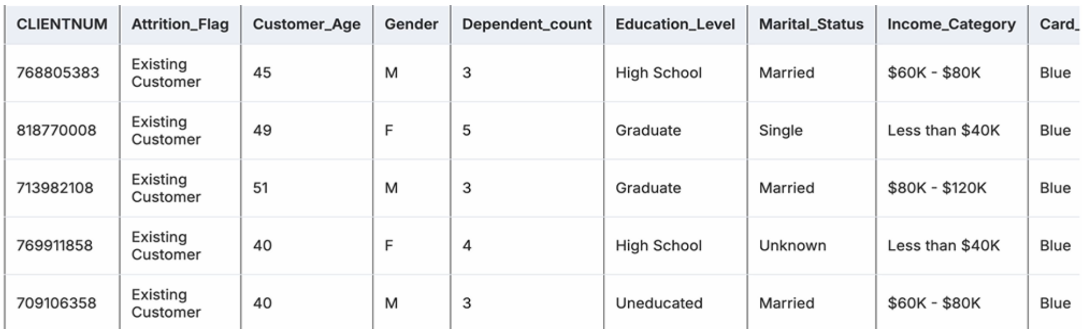
Predicting customer churn is crucial for any bank's profitability. We have developed a service giving you direct access to several machine learning models specifically trained to predict on whether customers will churn or not. Not only this service uses these models but also provides clear metrics on how well each performed during testing. Furthermore, you can view summaries of their predictive performance based on historical data.

2. Accessing the Churn Information

This service (API) is available online and can be accessed by visiting the following URL:

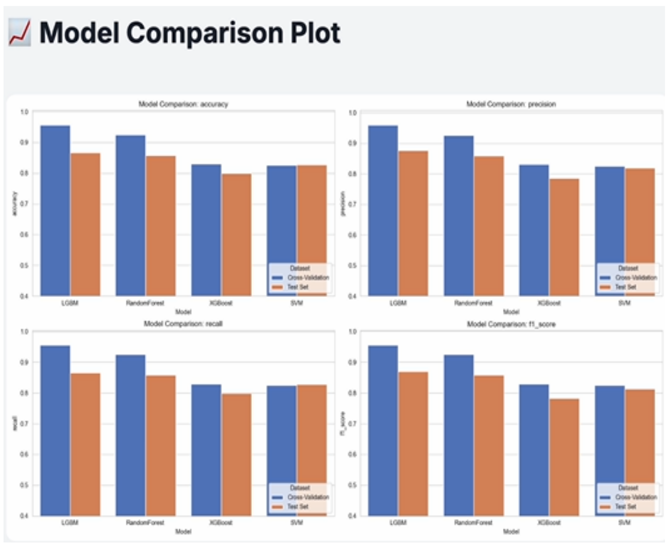
http://13.37.241.233:8000/

To provide essential context, the service displays the glimpse into the historical customer data that formed the basis for the prediction models.



CLIENTNUM	Attrition_Flag	Customer_Age	Gender	Dependent_count	Education_Level	Marital_Status	Income_Category	Card_
768805383	Existing Customer	45	M	3	High School	Married	$60K - $80K	Blue
818770008	Existing Customer	49	F	5	Graduate	Single	Less than $40K	Blue
713982108	Existing Customer	51	M	3	Graduate	Married	$80K - $120K	Blue
769911858	Existing Customer	40	F	4	High School	Unknown	Less than $40K	Blue
709106358	Existing Customer	40	M	3	Uneducated	Married	$60K - $80K	Blue

The service then details the specific machine learning approaches trained and evaluated for predicting customer churn, namely LGBM, Random Forest, SVM, and XGBoost. Crucially, the performance summary for each of these models, quantifying how effectively they predicted churn on unseen historical data. This summary includes key indicators— Accuracy, F1 Score, Precision, and Recall—derived from both cross-validation evaluations and performance on the final test dataset.



2. MLOps Workflow Stages

The MLOps workflow streamlines the deployment of our churn predictor model. It begins with standard model experimentation and training, resulting in a trained machine learning model. This model is then saved in a consistent format (.pkl file format) into a designated models/ directory for standardized handling. Next, a FastAPI template wraps the saved model, creating a REST API interface served using Uvicorn. Importantly, all Python dependencies for both the model and API are pinned in a requirements.txt file to ensure a reproducible environment. This entire package – API code, models, and dependencies – is then containerized using a Dockerfile.api, creating a portable Docker image. This image is then, pushed to a central registry (Docker Hub), making it accessible for deployment. Finally, a defined procedure is used to pull this image onto a cloud platform, AWS EC2, which includes infrastructure setup (instance provisioning, network configuration), resulting in a running, accessible API service.