**MINISTRY OF EDUCATION**

**FPT UNIVERSITY**



**MLFLOW CLASSIFICATION PROJECT DOCUMENTATION**

Class: **MSE.20HCM**

Subject: **AI in Production DevOps, DataOps, MLOps**

Instructor: **Huynh Cong Viet Ngu**

Reporter: **Hoang Thi Hoai Thuong**

HO CHI MINH CITY, 2025

**MLflow Classification Project Documentation**

## **1. Project Overview**

This project showcases a comprehensive machine learning workflow for a binary classification task using MLflow. It encompasses data generation, model training, hyperparameter tuning, model evaluation, registration, and deployment through a Flask web application. The project is containerized with Docker and deployed on AWS EC2 instances, utilizing an RDS PostgreSQL database for metadata storage and an S3 bucket for model artifacts.

## **2. Machine Learning Model**

### **2.1 Model Description**

* **Purpose**: Performs binary classification on synthetic data.
* **Algorithm**: Logistic Regression, a robust linear model for binary classification.
* **Functionality**: Predicts binary outcomes (0 or 1) based on 20 input features.

### **2.2 Dataset**

* **Source**: Synthetic dataset generated using sklearn.datasets.make\_classification.
* **Total Samples**: 1,000 (X: features, y: labels).
* **Features**: 20.
* **Class Distribution**:
  + y=0: 500 samples.
  + y=1: 500 samples.
* **Details**: View at <http://13.212.216.186:8001/experiment-summary>.

### **2.3 Training Classifier (classifier.py)**

The classifier.py script handles the following:

* **Data Generation**: Creates a synthetic dataset (1,000 samples, 20 features, 2 classes).
* **Data Splitting**: Allocates 80% for training and 20% for testing.
* **Model Training**: Trains a Logistic Regression model with varied hyperparameters.
* **Hyperparameter Tuning**: Tests combinations of C, solver, and tol.
* **Evaluation**: Assesses performance using accuracy and F1-score.
* **Model Logging**: Logs parameters, metrics, and models to MLflow.
* **Model Registration**: Registers the model with the highest F1-score as BestClassificationModel.

### **2.4 Hyperparameters**

* **C (Inverse of Regularization Strength)**:
  + Values: [0.1, 1.0, 10.0].
  + Impact: Smaller values increase regularization to prevent overfitting; larger values prioritize fitting training data.
* **Solver**:
  + Values: ['lbfgs', 'liblinear'].
  + Impact: 'lbfgs' is efficient for medium datasets; 'liblinear' suits smaller datasets.
* **Tolerance (tol)**:
  + Values: [1e-4, 1e-3].
  + Impact: Smaller values demand stricter convergence; larger values reduce computation time.
* **Total Runs**: 3 (C) × 2 (solver) × 2 (tol) = 12.

### **2.5 MLflow Integration**

* **Purpose**: Tracks experiments, logs parameters/metrics, stores models, and manages the model lifecycle.
* **Implementation**:
  + **Tracking URI**: SQLite (sqlite:///mlflow.db) for local testing; PostgreSQL on AWS RDS for production.
  + **Experiment**: Named classification\_experiment to organize runs.
  + **Logging**: Captures hyperparameters (C, solver, tol), metrics (accuracy, f1\_score), and models.
  + **Model Registration**: Registers the best model as BestClassificationModel.
  + **Comparison**: Displays a table of run metrics (run\_id, C, solver, tol, accuracy, f1\_score).

### **2.6 Model Comparison**

* Models are compared based on accuracy and f1\_score across hyperparameter combinations.
* Output: A table summarizing performance metrics for each run (viewable in MLflow UI or experiment summary).

## **3. Flask Web Application (app.py)**

### **3.1 Functionality**

* **Model Loading**: Dynamically loads the latest version of BestClassificationModel from MLflow.
* **Web Interface**:
  + **GET**: Renders index.html for inputting 20 features.
  + **POST**: Processes features, predicts outcomes, and displays results.
* **API Endpoint (/api/predict)**:
  + Accepts JSON with 20 features.
  + Returns JSON with prediction (0 or 1) or error.
* **Error Handling**: Manages model loading or prediction errors gracefully.

### **3.2 Interface**

* Accessible at [http://13.212.216.186:8001](http://13.212.216.186:8001/).
* Provides a user-friendly form for feature input and displays predictions.

## **4. Deployment Setup**

### **4.1 Local Docker Configuration**

* **Dockerfile**:
  + **Base Image**: python:3.11-slim.
  + **Steps**:
    - Installs dependencies from requirements.txt.
    - Copies app.py, classifier.py, templates/, and start.sh.
    - Runs classifier.py to train/register the model.
    - Exposes ports 8001 (Flask) and 5000 (MLflow).
    - Executes start.sh to start MLflow and Flask.
* **Steps**:
  + Clone repository:git clone <https://github.com/ThuongHoang456189/mlflow-flask-example.git>cd mlflow-flask-example
  + Build image:docker build -t mlflow-classifier-flask-app-included .
  + Run container:docker run -p 8001:8001 -p 5000:5000 mlflow-classifier-flask-app-included
  + Access Flask at <http://localhost:8001> and MLflow at <http://localhost:5000>.

### **4.2 AWS Deployment Architecture**

* **Components**:
  + **EC2 Instance 1**: Hosts MLflow tracking server (<http://54.254.217.150:5000>).
  + **EC2 Instance 2**: Runs Flask app (<http://13.212.216.186:8001>).
  + **RDS PostgreSQL**: Stores MLflow metadata.
  + **S3 Bucket**: Stores artifacts (e.g., BestClassificationModel).

#### **4.2.1 Prerequisites**

* **EC2 Instances**: Two t2.micro instances (free tier):
  + Configure ports: 5000 (MLflow), 8001 (Flask), 22 (SSH).
  + Install make, python, python-venv via SSH.
* **RDS PostgreSQL**: Configured for access by both EC2 instances.
* **S3 Bucket**: Allows EC2 instances to store artifacts.

#### **4.2.2 EC2 Instance 1 (MLflow Server)**

* **Source Files** (mlflow-flask-example/aws-deploy/mlflow-server):
  + classifier.py: Modified to use RDS environment variables.
  + Makefile: Automates MLflow server deployment.
  + requirements.txt: Lists MLflow dependencies.
* **Makefile Summary**:
  + **Variables**:
    - MLFLOW\_TRACKING\_URI: <http://localhost:5000>.
    - S3\_BUCKET\_NAME, RDS\_ENDPOINT, RDS\_PORT, RDS\_USERNAME, RDS\_PASSWORD, RDS\_DATABASE: AWS credentials.
    - VENV\_DIR: venv.
  + **Targets**:
    - all: Runs setup, venv, install, mlflow-server, run.
    - setup: Displays environment variables.
    - venv: Creates/verifies virtual environment.
    - install: Installs dependencies, checks for curl.
    - mlflow-server: Runs MLflow server with 2 workers, logs to mlflow\_server.log.
    - run: Executes classifier.py.
    - clean: Removes environment/logs.
    - help: Lists targets.
  + **Features**:
    - Uses /bin/bash.
    - Integrates S3 and RDS.
    - Includes error handling and logging.
* **Deployment Steps**:
  + Copy files to EC2:scp -i "MLflow-Classifier-Server-SSH-key.pem" Makefile [ubuntu@54.254.217.150:/home/ubuntu/mlflow-src](mailto:ubuntu@54.254.217.150:/home/ubuntu/mlflow-src)
  + Run:make all
  + Access MLflow at <http://54.254.217.150:5000>.

#### **4.2.3 EC2 Instance 2 (Flask Server)**

* **Source Files** (mlflow-flask-example/aws-deploy/flask-server):
  + templates/: HTML files (index.html, error.html, experiment-summary.html).
  + app.py: Flask app entry point.
  + Dockerfile: Builds hoangthuongdev/mlflow-classifier-flask-app:latest.
  + Makefile: Automates Docker deployment.
  + requirements.txt: Flask dependencies.
* **Makefile Summary**:
  + **Variables**:
    - IMAGE\_NAME: hoangthuongdev/mlflow-classifier-flask-app:latest.
    - CONTAINER\_PORT, HOST\_PORT: 8001.
  + **Targets**:
    - all: Runs install-docker, pull-image, run-container.
    - install-docker: Installs Docker.
    - pull-image: Pulls image from Docker Hub.
    - run-container: Runs container in detached mode.
    - debug-container: Debugs with /bin/bash (host network).
    - debug-container-bridge: Debugs with bridge network.
    - stop-container, clean: Cleanup tasks.
    - help: Lists targets.
  + **Features**:
    - Supports host/bridge networking.
    - Includes debugging and error handling.
* **Deployment Steps**:
  + Build and push image:docker build -t mlflow-classifier-flask-app .  
    docker push hoangthuongdev/mlflow-classifier-flask-app:latest
  + Copy Makefile to EC2:scp -i "MLflow-Flask-Server-SSH-key.pem" Makefile [ubuntu@13.212.216.186:/home/ubuntu/mlflow-example](mailto:ubuntu@13.212.216.186:/home/ubuntu/mlflow-example)
  + Run:make all
  + Access Flask at <http://13.212.216.186:8001>.

## **5. Conclusion**

This project delivers a robust machine learning pipeline using MLflow for experiment tracking and model management, paired with a Flask web application for user interaction. The AWS deployment leverages EC2 for compute, RDS for metadata, and S3 for artifacts, ensuring scalability and reliability. Docker containerization guarantees consistency across environments, making the project portable and maintainable.