Student Performance Prediction System

1. Project Overview

This project implements a complete Machine Learning lifecycle to predict student academic performance using multiple regression models. The system covers data processing, model training, evaluation, selection of best model, containerization, and cloud deployment on AWS. The entire workflow is automated using Continuous Integration and Continuous Deployment pipelines.

2. Problem Statement

Educational institutions require predictive analytics to understand student academic outcomes. This system predicts student exam performance based on relevant features such as prior academic scores, study related inputs, parental background information, and other structured attributes.

3. Model Training Pipeline

The training pipeline performs the following steps:

1. Load cleaned dataset and split into training and testing sets
2. Train multiple regression models including Linear Regression, Decision Tree, Random Forest, Gradient Boosting, and XGBoost
3. Evaluate each model using R2 score metric
4. Select the model with highest R2 score automatically
5. Save best performing model as artifacts model.pkl

4. Model Selection Strategy

Each trained model is evaluated on test data. R2 score is used as the primary evaluation metric. The model with the highest R2 score is selected automatically. This approach ensures that the best performing model is used in production without manual intervention.

5. Application Layer

The saved model is loaded inside the application layer. Users provide input features through an interface or API endpoint. The application processes inputs and returns predicted student performance scores in real time.

6. Docker Containerization

The application is containerized using Docker. The Docker image includes project source code, trained model artifact, and required dependencies. This ensures consistency across development and production environments.

7. Continuous Integration and Delivery

GitHub Actions is used to automate the build and deployment workflow.

1. Code push to main branch triggers workflow
2. Continuous Integration job validates repository
3. Docker image is built automatically
4. Image is pushed to Amazon Elastic Container Registry
5. Self hosted EC2 runner pulls latest image
6. Container is restarted with updated model

8. AWS Infrastructure

The deployment architecture includes the following AWS services:

1. Amazon EC2 for hosting Docker container
2. Amazon Elastic Container Registry for storing Docker images
3. IAM for access control and security management

9. Deployment Flow Summary

Developer pushes updated code. GitHub workflow builds and pushes Docker image to Amazon ECR. EC2 runner pulls latest image. Docker container runs updated prediction service. Application becomes live automatically.

10. Future Improvements

1. Reduce Docker image size through optimized base image
2. Migrate deployment to Amazon ECS for container orchestration
3. Add monitoring and logging integration
4. Implement automatic model retraining pipeline
5. Add data drift detection mechanisms

Author Credit

Prepared by Satyam Gajjar. Machine Learning Engineering and Cloud Deployment Project Documentation.