MLOps Group No: 92

Group Members Names:

1. PEYALA SAMARASIMHA REDDY - 2023AA05072
2. PEGALLAPATI SAI MAHARSHI - 2023AA05924
3. ANIRUDDHA DILIP MADURWAR - 2023AA05982
4. K VAMSIKRISHNA - 2023AA05209

MLOps Assignment-1 Summary

M1: MLOps Foundations

Objective:
The objective of this task was to set up a simple CI/CD pipeline and implement version control practices using Git, as well as use a CI/CD tool (GitHub Actions in this case) to automate machine learning workflows.

Tasks Completed:

1. **Set Up a CI/CD Pipeline:**
   We used **GitHub Actions** to create an automated pipeline for the Titanic survival prediction model. The pipeline included three essential stages:
   - **Linting:** We implemented a Python linting stage using `pylint` to check the code for errors and ensure it adheres to standard coding practices.
   - **Testing:** We incorporated a testing stage to run the model training script (`train.py`) to ensure it functions correctly, with scope for adding additional unit tests in the future.
   - **Deployment:** We automated the deployment of the trained model, saving it as a `.pkl` file in the `models/` directory. Additionally, we used **GitHub Actions' upload artifact** feature to store the trained model as an artifact for later use.
2. **Justification:**
   Using GitHub Actions is a simple and effective way to automate CI/CD pipelines for machine learning projects. It provides out-of-the-box support for Python, making it easy to set up automated stages for linting, testing, and deployment without additional configurations.
3. **Version Control:** We implemented **Git** for version control of the project. Key activities included:

    - **Branching and Merging:** We used Git branches to develop features and then merged them into the `main` branch. This ensured a clean main branch and allowed experimentation in separate branches.
    - **Pull Requests (PRs):** We created pull requests to review changes before merging them into the main branch, ensuring code quality and collaboration.
4. **Justification:**
   Git provides an efficient way to manage versions of code and collaborate within teams. Branching allows experimentation without affecting the main codebase, while pull requests ensure quality control through peer reviews.

M2: Process and Tooling

Objective:
This task focused on gaining hands-on experience with MLOps tools like **MLflow** for experiment tracking and **DVC** (Data Version Control) for dataset versioning, helping to streamline machine learning workflows.

Tasks Completed:

1. **Experiment Tracking with MLflow:**
   We integrated **MLflow** into our machine learning workflow to track the experiments for model training. We recorded metrics like accuracy, hyperparameters, and results of **three different ML algorithms with three different settings, total 9 training runs**.
    - Each training run was executed with different parameters for the Random Forest Classifier.
    - We used MLflow's logging capabilities to track the **accuracy** and **parameters** of each run, making it easy to compare results and identify the best model configuration in ML flow UI.
2. **Justification:**
   MLflow simplifies experiment tracking by providing a central repository for metrics, parameters, and model artifacts. This helps ensure reproducibility and allows for easy comparison between various model configurations.
3. **Data Versioning with DVC:**
   We implemented **DVC** to version control the dataset used in the Titanic model. DVC helped to keep track of different versions of the dataset and enabled easy reversion to previous versions. We also used DVC to link data files with Git, ensuring that the dataset versions were synchronized with the model code.
    - We recorded the current version of the Titanic dataset used in training, made changes to data by adding one row at bottom.
    - We demonstrated how to revert to a previous dataset version using DVC's checkout and pull features.
4. **Justification:**
   DVC is specifically designed for data versioning and helps manage large datasets that

cannot be easily handled by Git alone. It ensures that data versions are tightly coupled with the code, providing a robust way to track both datasets and models.

M3: Model Experimentation and Packaging

Objective:
This stage focused on hyperparameter tuning for the machine learning model, as well as packaging the trained model for deployment using Docker and Flask.

Tasks Completed:

1. **Hyperparameter Tuning with Optuna:**
   We used **Optuna** to perform hyperparameter tuning on the **Random Forest Classifier**. We experimented with four key hyperparameters:
   - **n_estimators** (number of trees)
   - **max_depth** (maximum depth of the tree)
   - **min_samples_split** (minimum samples required to split a node)
   - **min_samples_leaf** (minimum samples required at a leaf node)
2. Optuna's optimization process helped us find the best hyperparameters, resulting in improved accuracy. We documented the tuning process and shared the best hyperparameters found:
   - **Best hyperparameters:** `n_estimators=70`, `max_depth=18`, `min_samples_split=8`, `min_samples_leaf=1`
3. **Justification:**
   Hyperparameter tuning is critical for improving model performance, and Optuna provides an efficient and automated way to optimize multiple hyperparameters simultaneously, saving both time and computational resources.
4. **Model Packaging with Docker and Flask:**
   We packaged the best-performing model using **Docker** and created a simple **Flask** application to serve the model as an API.
   - **Dockerfile:** We wrote a Dockerfile that installed all the necessary dependencies and bundled the trained model into a Docker container.
   - **Flask Application:** The Flask application exposes an endpoint where users can send data to get predictions from the trained Titanic model. We created a index.html UI to directly enter inputs and predicts outputs and also we used this endpoint to predict results in thunder client.
5. **Justification:**
   Docker is widely used for packaging machine learning models because it provides an isolated and reproducible environment for running models. Flask allows easy deployment of machine learning models as APIs, enabling them to be consumed by other systems.