# ML Ops Assigment-1

Team Contribution:

1. JAIDEEP SINGH SACHDEV (2023aa05196): 100%

2. VIKRAM S (2023aa05628): 100%

3. SHAKTI NATH SAINI (2023aa05387): 100%

4. CHOPRA CHIRAG SURESH SANGEETA (2023aa05520): 100%

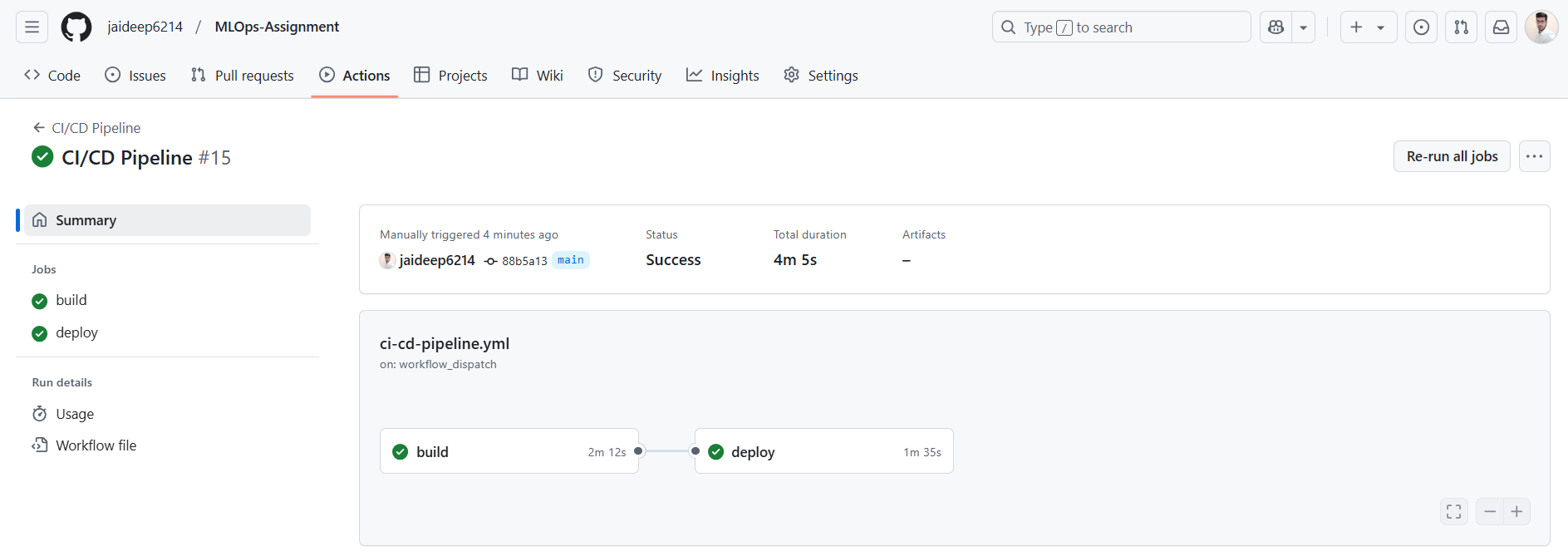
**M1: MLOps Foundations**

**Objective**:  
Understand the basics of MLOps and implement a simple CI/CD pipeline.

**Deliverables:**

* **Report Detailing the CI/CD Pipeline Stages**:
* **Linting**:  
  **Purpose**: The linting stage ensures that the code adheres to consistent style guidelines and is free of basic errors before being tested or deployed. This improves the readability and quality of the code.  
  **Implementation**: In this stage, I used a Python linter such as flake8 or pylint to check for potential issues such as unused imports, syntax errors, and incorrect formatting.  
  The pipeline runs flake8 on every commit to ensure that all code changes meet the defined standards.
* **Testing**:  
  **Purpose**: The testing stage ensures that the machine learning model and its associated code work as expected. It runs unit tests and integration tests to verify the correctness of the code.  
  **Implementation**: I integrated a testing tool, such as pytest, to automatically run the tests for the machine learning model each time a change is made. This helps catch any bugs or regressions early in the development process.  
  After running the tests, the pipeline reports whether all tests have passed or failed.
* **Deployment**:  
  **Purpose**: The deployment stage automates the process of moving the model from the development environment to a production environment. This ensures that the model is deployed quickly and reliably.  
  **Implementation**: Once the linting and testing stages are successful, I set up the deployment stage to automatically deploy the trained model to a cloud platform, such as AWS S3, or a containerization platform like **Docker** or **Kubernetes**. This stage ensures that the latest version of the model is always available for use in production.
* **Pipeline Workflow:**

1. The pipeline is triggered on **push** events, meaning that each time a code change is pushed to the repository, the pipeline automatically starts.
2. The first step is **linting**, followed by **testing**, and finally **deployment** if the tests pass.
3. If any step fails, the pipeline will stop, and the team will be notified, ensuring that faulty code doesn’t make it into production.

* **Screenshots or Logs Showing Successful Runs of the Pipeline**:  
  
* A screenshot of a computer

  AI-generated content may be incorrect.
* A screenshot of a computer

  AI-generated content may be incorrect.
* **Git Repository Link with Branches and Merge History**:  
  https://github.com/jaideep6214/MLOps-Assignment

**M2: Process and Tooling**

**Objective**: Gain hands-on experience with popular MLOps tools and understand the processes they support.

**Deliverables:**

**MLflow experiment logs with different runs and their results.**

**A screenshot of a computer

AI-generated content may be incorrect.**

**A screenshot of a computer

AI-generated content may be incorrect.**

**A screenshot of a computer

AI-generated content may be incorrect.**

**A DVC repository showing different versions of the dataset.**

**A screenshot of a computer program

AI-generated content may be incorrect.**

**M3: Model Experimentation and Packaging**

**Objective**: Train a machine learning model, perform hyperparameter tuning, and package the model for deployment.

**Deliverables**:

**A report on hyperparameter tuning results.**

**Objective:  
This report aims to outline the process and results of hyperparameter tuning for the Random Forest Classifier model using GridSearchCV. The goal is to improve the model’s performance on the Titanic dataset by optimizing hyperparameters.**

**Dataset and Preprocessing:**

* **Dataset: The Titanic dataset, which is processed and saved in data/processed/titanic.csv.**
* **Features: The features of the dataset include information such as the passenger's age, sex, class, and other factors that could influence survival.**
* **Target Variable: The target variable is "Survived," indicating whether a passenger survived (1) or not (0).**

**Model Selection:**

**The selected model is the Random Forest Classifier, an ensemble learning method that builds multiple decision trees and combines their predictions to improve accuracy and reduce overfitting.**

* **RandomForestClassifier is initialized with a random state of 42 for reproducibility.**

**Hyperparameter Tuning:**

**To tune the hyperparameters of the Random Forest model, GridSearchCV was used. The grid search method exhaustively tests a specified range of hyperparameters for the model and identifies the best combination based on cross-validation performance.**

**Hyperparameters Considered:**

**The following hyperparameters were tuned:**

* **n\_estimators: The number of trees in the forest. Values tested: [50, 100, 200].**
* **max\_depth: The maximum depth of the trees. Values tested: [None, 10, 20, 30].**
* **min\_samples\_split: The minimum number of samples required to split an internal node. Values tested: [2, 5, 10].**

**GridSearchCV Implementation:**

* **Cross-Validation: 5-fold cross-validation was used to assess the performance of the model with different combinations of hyperparameters.**
* **Parallel Processing: The n\_jobs=-1 parameter enabled parallel processing, which speeds up the grid search by using all available CPU cores.**
* **Verbose: The verbose=2 parameter provided detailed logs during the grid search process.**

**Model Evaluation:**

**After the hyperparameter tuning was completed, the best model was identified using grid\_search.best\_estimator\_.**

* **The model was evaluated using the accuracy metric on the test set.**
* **The accuracy of the best model was found to be 0.8379 (approximately 84%).**

**This result indicates that the tuned Random Forest Classifier model achieved an accuracy of 83.8% on the test data, which is a strong performance.**

**MLflow Integration:**

**To track and log the experiment, MLflow was used. The following aspects were logged:**

* **Parameters: The best hyperparameters identified during the grid search.**
* **Metrics: The accuracy of the best model on the test data.**
* **Model: The trained model was saved and logged in MLflow for later use.**

**Model Saving:**

**The best model was saved as a Pickle file for future use**

**Metrics:**

**The performance of the model was evaluated based on accuracy, and the results were saved in a JSON file.**

**{**

**"accuracy": 0.8379888268156425**

**}**

**The model achieved an accuracy of approximately 83.8%, demonstrating that the hyperparameter tuning process effectively improved the model's performance.**

**A Dockerfile and Flask application code.**

**DockerFile:**

**A screenshot of a computer code

AI-generated content may be incorrect.**

**Docker Compose:**

**A screenshot of a computer code

AI-generated content may be incorrect.**

**Flask Code:  
A screenshot of a computer program

AI-generated content may be incorrect.**

**Screenshots of the model running in a Docker container.**

A screenshot of a computer

AI-generated content may be incorrect.

**Summary of Work Completed: Hyperparameter Tuning for Random Forest Classifier**

**Description of Work Completed:**

In this project, I focused on optimizing the performance of a **Random Forest Classifier** by performing **hyperparameter tuning** using **GridSearchCV**. The objective was to improve the model's accuracy in predicting the survival of passengers from the Titanic dataset.

The dataset was split into training and testing sets, and the Random Forest model was trained on the training data. A set of hyperparameters, including n\_estimators, max\_depth, and min\_samples\_split, were tuned using GridSearchCV. Cross-validation was employed to evaluate the model's performance across different combinations of hyperparameters.

The tuned model achieved an accuracy of **83.8%** on the test dataset, which is a strong performance. The model was saved and logged using **MLflow** for version tracking and reproducibility.

**Choices Made and Justifications:**

1. **Random Forest Classifier**:
   * The **Random Forest Classifier** was chosen for its robustness, flexibility, and ability to handle high-dimensional datasets without requiring too much fine-tuning. It is also effective for classification problems, especially with structured data like the Titanic dataset.
2. **GridSearchCV for Hyperparameter Tuning**:
   * **GridSearchCV** was chosen because it performs an exhaustive search over a specified hyperparameter grid. This method allows for thorough exploration of multiple hyperparameter values, which is important for achieving the best possible performance from the model.
3. **Hyperparameters Tuned**:
   * **n\_estimators**: The number of trees in the forest was varied (50, 100, 200) to check for the optimal number. More trees can improve model performance but may also lead to higher computational costs.
   * **max\_depth**: Varying the tree depth (None, 10, 20, 30) helps prevent overfitting and underfitting. Limiting depth prevents the model from becoming too complex and overfitting the training data.
   * **min\_samples\_split**: This hyperparameter controls the minimum number of samples required to split an internal node. By testing values of 2, 5, and 10, I aimed to balance model complexity and overfitting.
4. **MLflow for Experiment Tracking**:
   * **MLflow** was used to log the hyperparameters, metrics (accuracy), and model. This decision was made to ensure reproducibility and to provide a centralized system for tracking experiments and model versions.
5. **Evaluation Metric**:
   * **Accuracy** was chosen as the evaluation metric because it provides a clear measure of the model’s overall performance. Since the dataset was relatively balanced, accuracy was a suitable metric to assess the model's effectiveness.