

MLOps Project Report

Yunhang Chi, Wenxi Huang, Shiyu Mao, Zhongke Sun

April 9, 2025

1 Introduction

This report presents an MLOps pipeline built around the classic iris classification problem. The goal of this project was to demonstrate a practical approach to model training, evaluation, deployment, and basic versioning within a simplified Python ecosystem. Although the iris dataset is relatively small and straightforward, it offers a convenient framework for exploring key concepts like reproducible training, containerization, and command-line automation.

2 Workflow Design

A typical workflow begins by calling the training step. The user can install dependencies from `requirements.txt` and then invoke `python model.training.py` to produce a trained random forest model. If they wish to evaluate performance again or modify certain parameters, they can adjust the training script and re-run the process. For inference, `python model.prediction.py` loads the saved model and displays predictions for a sample input. Once satisfied with local results, the user can containerize the project. The project includes a `Dockerfile` that sets up a minimal Python environment, installs the required libraries, and defaults to running the model code upon container startup. With the Docker image built, `CLI.py` offers straightforward commands like `build`, `push`, `pull`, and `run_model` for sharing or launching the container.

3 Project Structure

The project follows a modular design to keep training, evaluation, inference, and data preprocessing tasks both transparent and extensible. Each stage is defined in a separate Python script:

preprocessing.py Loads and prepares the iris dataset for training and testing. It also includes an optional exploratory analysis function that generates basic visualizations such as scatterplots and correlation matrices.

model_training.py Implements a `train_model()` function that trains a random forest classifier, prints training duration, and saves the final model to a local file (`model.pkl`). Once the model is trained, the script triggers an evaluation routine to measure classification performance.

model_evaluation.py Defines `evaluate_model(...)`, which computes metrics like accuracy, precision, recall, and F1 score. It also plots and saves a confusion matrix. The script includes per-class accuracy reports, serving as a basic illustration of how fairness or bias checks might be performed in a more complex scenario.

model_prediction.py Loads the persisted model file and demonstrates a simple inference process. In this example, it predicts the species of a single test instance, though the logic can be adapted to support batch or interactive prediction.

model_registry.py Illustrate a simple versioning workflow. Each time the `register_model()` function is called, it increments the version by counting existing “Version” lines in the registry log, logs metadata (path and timestamp) to `model_registry.txt`, and appends a “Version X” header and a separator line for clarity.

CLI.py Handles both container operations and local model tasks. Users can build, run, push, or pull a Docker image, and can also train or predict locally without invoking Docker.

A Appendix: Screenshots

```
C:\Users\hwx1\Desktop\MLops\docker_display\docker_deploy 2\docker_deploy>python CLI.py build
[+] Building 398.9s (11/11) FINISHED
=> [internal] load build definition from Dockerfile
=> transferring dockerfile: 198B
=> [internal] load metadata for docker.io/library/python:3.11-slim
=> [auth] library/python:pull token for registry1.docker.io
=> [internal] load .dockerignore
=> transferring context: 2B
=> [1/5] FROM docker.io/library/python:3.11-slim@sha256:881075da77b2b55c23c888251026fb69a7b2bf92471e491f
=> resolve docker.io/library/python:3.11-slim@sha256:881075da77b2b55c23c888251026fb69a7b2bf92471e491f
=> [internal] load build context
=> transferring context: 7.88kB
=> CACHED [2/5] WORKDIR /app
=> CACHED [3/5] COPY requirements.txt requirements.txt
=> [4/5] COPY . .
=> [5/5] RUN pip install --no-cache-dir -r requirements.txt
=> exporting to image
=> exporting layers
=> exporting manifest sha256:7891e23b23d3db384f4fb9b2dc4c9f7385c180e975b78980bf6d98f4d6e685
=> exporting config sha256:1485e1c94b9948189ad8c1e47c8517bc7b4f40419e1b72a69818b015bad759f
=> exporting attestation manifest sha256:891c7597f5e5c8c7bb24d5b2a9648114ba8f6b8d3ad602f740f67b487697
=> exporting manifest list sha256:1a649fdbelcc3b12daeb5c6856bd59533effcd126cda0edc91c1262dd027b23a
=> naming to docker.io/wenxi1203/mlops-iris:v1
=> unpacking to docker.io/wenxi1203/mlops-iris:v1
```

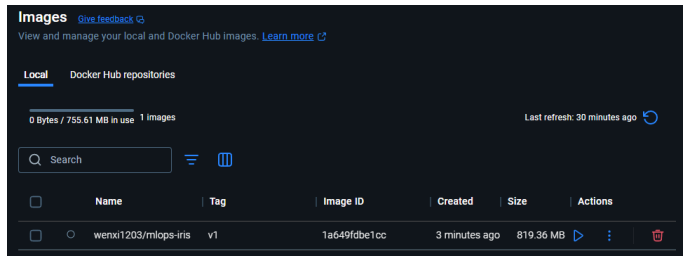


Figure 1: Build

```
C:\Users\hwx1\Desktop\MLops\docker_display\docker_deploy 2\docker_deploy>python CLI.py run_container
Model training completed in 0.13 seconds.
Evaluation Metrics:
Accuracy : 1.00
Precision: 1.00
Recall : 1.00
F1 Score : 1.00
Confusion Matrix:
[[10 0 0]
 [0 0 0]
 [0 0 1]]
Confusion matrix saved as confusion_matrix.png
Per-Class Accuracy:
Class 0 accuracy: 1.00
Class 1 accuracy: 1.00
Class 2 accuracy: 1.00
Model saved as model.pkl
Registered model version 1
log saved to: C:\Users\hwx1\Desktop\MLops\docker_display\docker_deploy 2\docker_deploy\model_registry.txt
Model loaded from model.pkl
Prediction for [[6.1, 2.8, 4.7, 1.2]]: [1]
True label: 1
```

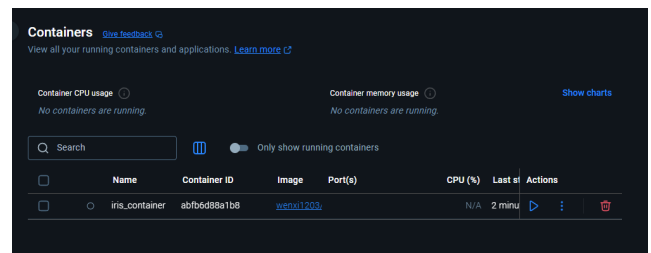


Figure 2: Run Container

```
C:\Users\hwx1\Desktop\MLops\docker_display\docker_deploy 2\docker_deploy>python CLI.py push
Authenticating with existing credentials... [Username: wenxi1203]

Info -> To login with a different account, run 'docker logout' followed by 'docker login'

Login Succeeded
The push refers to repository [docker.io/wenxi1203/mlops-iris]
edba8d8f0439: Pushed
922ce065dc70: Pushed
4046b1a5d574c: Pushed
8a628cdd7ccc: Pushed
f6b3bbd679a5: Pushed
dd5932d74b25: Pushed
2baa8306c37d: Pushed
100949e89857: Pushed
7526bac51f84: Pushed
v1: digest: sha256:1a649fdbelcc3b12daeb5c6856bd59533effcd126cda0edc91c1262dd027b23a size: 856
```

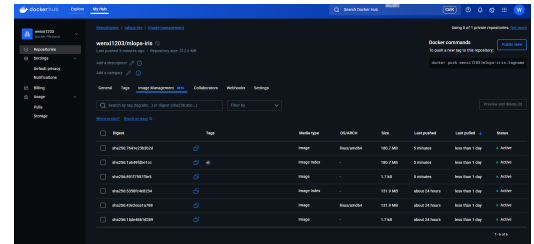


Figure 3: Push Docker Hub

```
C:\Users\hwx1\Desktop\MLops\docker_display\docker_deploy 2\docker_deploy>python CLI.py stop
Stopping container: iris
=> stopping container: iris
=> removing container: iris
=> deleting image: wenxi1203/mlops-iris:v1
=> deleting image: wenxi1203/mlops-iris
```

```
C:\Users\hwx1\Desktop\MLops\docker_display\docker_deploy 2\docker_deploy>python CLI.py pull
v1: Pulling from wenxi1203/mlops-iris
Digest: sha256:1a649fdbelcc3b12daeb5c6856bd59533effcd126cda0edc91c1262dd027b23a
Status: Image is up to date for wenxi1203/mlops-iris:v1
docker.io/wenxi1203/mlops-iris:v1
```

Figure 4: Delete Figure and Pull

```
C:\Users\hwx1\Desktop\MLops\docker_display\docker_deploy 2\docker_deploy>python CLI.py run_model
Model training completed in 0.07 seconds.
Evaluation Metrics:
Accuracy : 1.00
Precision: 1.00
Recall : 1.00
F1 Score : 1.00
Confusion Matrix:
[[10 0 0]
 [0 9 0]
 [0 0 11]]
Confusion matrix saved as confusion_matrix.png
Per-Class Accuracy:
Class 0 accuracy: 1.00
Class 1 accuracy: 1.00
Class 2 accuracy: 1.00
Model saved as model.pkl
Registered model version 1
Log saved to: C:\Users\hwx1\Desktop\MLops\docker_display\docker_deploy 2\docker_deploy\model_registry.txt
Model loaded from model.pkl
Prediction for [[6.1, 2.8, 4.7, 1.2]]: [1]
true label: 1
```

```
C:\Users\hwx1\Desktop\MLops\docker_display\docker_deploy 2\docker_deploy>python CLI.py train
Model training completed in 0.28 seconds.
Evaluation Metrics:
Accuracy : 1.00
Precision: 1.00
Recall : 1.00
F1 Score : 1.00
Confusion Matrix:
[[10 0 0]
 [0 9 0]
 [0 0 11]]
Confusion matrix saved as confusion_matrix.png
Per-Class Accuracy:
Class 0 accuracy: 1.00
Class 1 accuracy: 1.00
Class 2 accuracy: 1.00
Model saved as model.pkl
Registered model version 1
Log saved to: C:\Users\hwx1\Desktop\MLops\docker_display\docker_deploy 2\docker_deploy\model_registry.txt
```

Figure 5: Run and Train the Model

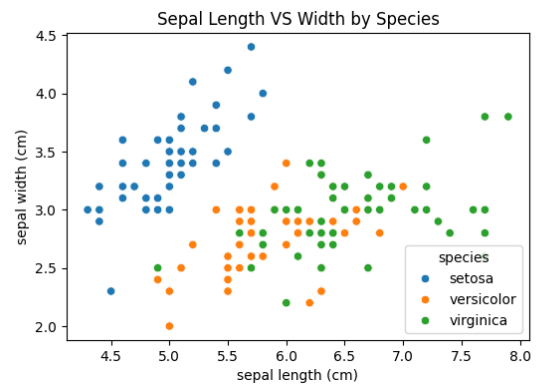
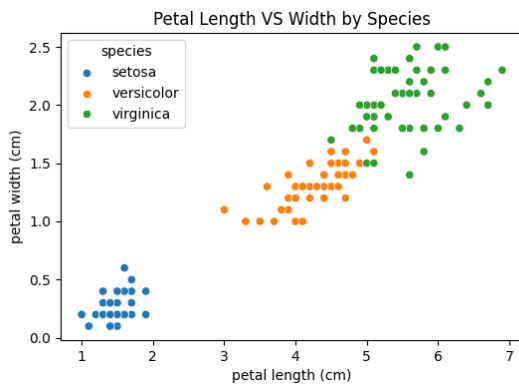
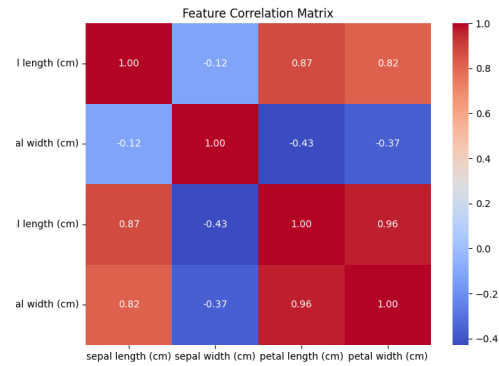
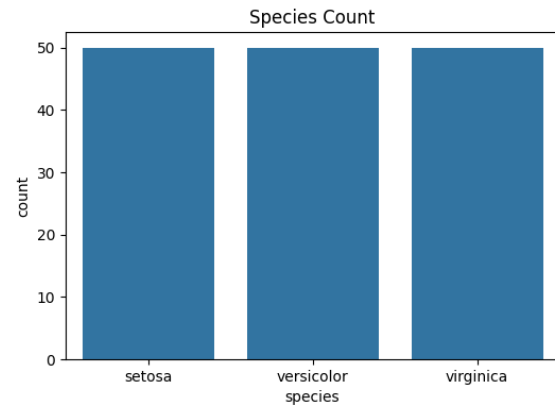


Figure 6: Visualization