MLOps Fundamentals

Implementing MLOps: A Practical Example

Overview

Now that you understand MLOps principles and tools, it's time to **apply them in a real-world scenario**. In this hands-on lesson, you'll modify an existing ML pipeline to incorporate **MLOps best practices**.

Learning Objectives

By the end of this microlesson, you will:

- **Analyze** an ML pipeline and identify missing MLOps components.
- **Modify** the pipeline to include experiment tracking, versioning, and automation.
- **Use MLflow** to log models and track performance over multiple runs.

1. Understanding the Existing Pipeline

You've been given a basic ML training script, but it lacks:

- **Experiment tracking** (no way to compare different runs).
- **Model versioning** (no structured model storage).
- **Automated deployment** (manual model saving/loading).

Before: Original ML Pipeline

Copy

```
import joblib
from sklearn.ensemble import RandomForestClassifier

# Load dataset
X_train, X_test, y_train, y_test = load_data()

# Train model
model = RandomForestClassifier()
model.fit(X_train, y_train)

# Save model
joblib.dump(model, "model.pkl")
```

2. Hands-On: Adding MLOps Best Practices

Step 1: Add Experiment Tracking with MLflow

Modify the script to log key parameters and metrics in **MLflow**.

Copy

```
import mlflow
import mlflow.sklearn

mlflow.start_run():
    model = RandomForestClassifier(n_estimators=100)
    model.fit(X_train, y_train)
    acc = model.score(X_test, y_test)
```

```
    mlflow.log_param("model_type", "RandomForest")
    mlflow.log_metric("accuracy", acc)
    mlflow.sklearn.log_model(model, "model")
```

✅ **Now, each run is logged, making it easier to compare models.**

Step 2: Implement Model Versioning

Modify `model_saving.py` to store models in the **MLflow Model Registry**.

Copy

```
mlflow.sklearn.log_model(model, "model")
mlflow.register_model("runs:/<run_id>/model", "MLPipelineModel")
```

✅ **Now, models are versioned and accessible via MLflow UI.**

Step 3: Prepare for Deployment

Modify `deploy.py` to load models directly from the **MLflow Model Registry**.

Copy

```
import mlflow.pyfunc
model = mlflow.pyfunc.load_model("models:/MLPipelineModel/Production")
```

✅ **Now, the model can be served dynamically without manual file handling.**

3. Key Takeaways

✅ MLOps **improves ML workflows** by adding automation and tracking.
✅ MLflow enables **experiment logging, model versioning, and easy deployment**.
✅ A structured pipeline ensures **scalability and maintainability**.

Bonus Challenge: Deploy Your Model for Public Access

Want to **share your model online**? Try deploying it using **FastAPI + Docker + Render (or Hugging Face Spaces)**.

Steps:

1. **Wrap your model in an API** using FastAPI:

Copy

```python
from fastapi import FastAPI
import mlflow.pyfunc

app = FastAPI()
model = mlflow.pyfunc.load_model("models:/MLPipelineModel/Production")

@app.post("/predict/")
async def predict(data: dict):
    return {"prediction": model.predict([data["input"]]).tolist()}
```

2. **Containerize it with Docker:**

Copy

```
docker build -t my-ml-api .
docker run -p 8000:8000 my-ml-api
```

3. **Deploy on Render, Hugging Face Spaces, or another cloud service.**

Once deployed, you can share your **public API endpoint** with others to test your model in real-time! 🎯

< Previous