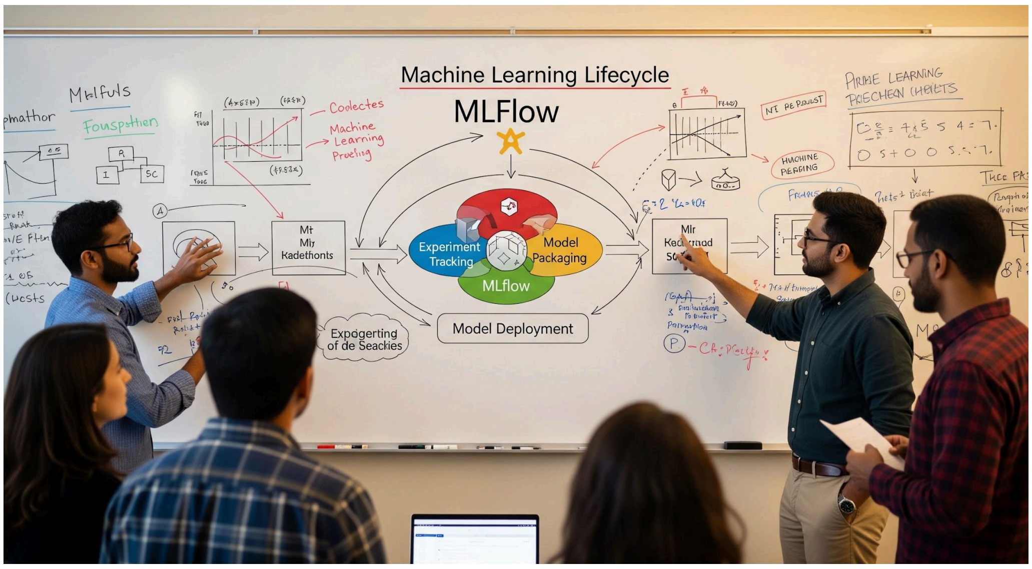
Day 4: MLFlow - Managing the Machine Learning Lifecycle



Section 1: Introduction to MLFlow

Theory

What is MLFlow?
MLFlow acts as a project manager for your machine learning work. It tracks experiments, organizes code, saves models, and deploys them, making it easier to manage projects from testing to production.

Why Use MLFlow?

- **Track Experiments**: Log parameters, metrics, and artifacts to compare model runs.
- **Reproduce Work**: Package code and dependencies for consistent results.
- **Deploy Models**: Move models to production environments like web servers.
- **Collaborate**: Share experiments and models with your team.

MLFlow Components:

- **Tracking**: Logs parameters (learning rate), metrics (accuracy), and artifacts (model files) with a web UI.
- **Projects**: Packages code and dependencies for reproducibility.
- **Models**: Standardizes model formats for deployment.
- **Registry**: Manages model versions and stages.

Benefits:

- Simplifies managing RL, forecasting, or vision projects.
- Works with scikit-learn, PyTorch, and TensorFlow.

Challenges:

- Initial setup in PyCharm may require terminal use.
- Advanced features (e.g., distributed deployment) need careful configuration.

Hands-on Exercise (0 Minutes)

- No hands-on in this section; focus on understanding MLFlow's role.

Section 2: Installation and Setup in PyCharm

Installation Steps:

1. Open PyCharm, create a new project (`MLFlowTraining`).
2. Go to `File > Settings > Project > Python Interpreter`.
3. Click `+`, search for `mlflow`, install it, and apply.

Verify installation in PyCharm's terminal:

```
mlflow --version
```

Start the MLFlow UI in the terminal:

```
mlflow ui --port 5000
```

    - Access at `http://localhost:5000` in your browser.
    - UI displays experiments, stored in the `mlruns` directory.
- Port conflicts: Use `--port 5001` if 5000 is occupied.

Section 3: MLFlow Tracking

Theory

What is MLFlow Tracking?
MLFlow Tracking is your experiment notebook. It logs parameters (e.g., learning rate), metrics (e.g., accuracy), and artifacts (e.g., plots) for each run, viewable in a web UI. It's great for comparing RL hyperparameter tuning or vision model experiments.

Key Features:

- **Automatic Logging**: `mlflow.autolog()` logs scikit-learn or PyTorch details.
- **Manual Logging**: Use `mlflow.log_param()`, `mlflow.log_metric()`, and `mlflow.log_artifact()` for custom control.
- **Experiments**: Group runs for organization.
- **UI**: Visualize and compare runs at `http://localhost:5000`.

How It Works:

1. Start a run: `mlflow.start_run()`.
2. Log data: `mlflow.log_param("lr", 0.001)`.
3. Train and log: `mlflow.log_metric("accuracy", 0.95)`.
4. End run: `mlflow.end_run()`.

Benefits:

- Tracks experiment history for reproducibility.
- Supports your existing workflows.

Hands-on Exercise

Objective: Use MLFlow Tracking to log experiments for a linear regression model on the California Housing dataset in PyCharm.

Dataset: California Housing (regression dataset, available via `sklearn.datasets.fetch_california_housing`).

Code:

```python
import mlflow
import mlflow.sklearn
from sklearn.datasets import fetch_california_housing
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error
import numpy as np

# Load dataset
data = fetch_california_housing()
X_train, X_test, y_train, y_test = train_test_split(data.data,
data.target, test_size=0.2, random_state=42)

# Automatic logging
mlflow.autolog()
with mlflow.start_run(run_name="auto_log_run"):
    model = LinearRegression()
    model.fit(X_train, y_train)
    y_pred = model.predict(X_test)
    mse = mean_squared_error(y_test, y_pred)
    print(f"Auto-logged MSE: {mse:.4f}")

# Manual logging
mlflow.set_experiment("California_Housing_Experiment")
with mlflow.start_run(run_name="manual_log_run"):
    # Log parameters
    learning_rate = 0.01  # Simulated parameter for demo
    mlflow.log_param("learning_rate", learning_rate)
    # Train model
    model = LinearRegression()
    model.fit(X_train, y_train)
    # Log metrics
    y_pred = model.predict(X_test)
    mse = mean_squared_error(y_test, y_pred)
    mlflow.log_metric("mse", mse)
    # Log model
    mlflow.sklearn.log_model(model, "linear_model")
    print(f"Manually logged MSE: {mse:.4f}")
```

Questions:

1. How do the automatic and manual logging runs appear in the UI?
2. What parameters and metrics were logged automatically?

Section 4: MLFlow Projects

Theory

What is an MLFlow Project?
An MLFlow Project is a portable package for your ML code, like a project folder you can share. It includes scripts, dependencies, and instructions, ensuring others can run it the same way.

Key Components:

- **MLproject File**: A YAML file with project name, environment, and entry points.
- **Dependencies**: Listed in `requirements.txt`.
- **Entry Points**: Commands like `python train.py` with parameters.

How It Works:

1. Create a directory with code, `MLproject`, and `requirements.txt`.
2. Run with `mlflow run <project_dir> -P alpha=0.5`.

Benefits:

- Ensures reproducibility across machines.
- Simplifies collaboration for RL or vision tasks.

Hands-on Exercise

Objective: Create and run an MLFlow Project for a linear regression model in PyCharm.

Steps:

1. Create a project directory `california_housing_project` in PyCharm:

`MLproject`:

```
name: california_housing
entry_points:
  main:
    parameters:
      alpha: {type: float, default: 0.5}
    command: "python train.py {alpha}"
```

`requirements.txt`:

```
mlflow==2.13.2
scikit-learn==1.5.0
numpy==1.26.4
```

`train.py`:

```
import mlflow
import mlflow.sklearn
from sklearn.datasets import fetch_california_housing
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error
import argparse
import sys

# Set tracking URI programmatically
mlflow.set_tracking_uri("http://localhost:5000")
```

```
parser = argparse.ArgumentParser(description="Train a linear
regression model")
parser.add_argument("alpha", type=float, help="Alpha parameter
for training")
args = parser.parse_args()

print(f"Running with Python: {sys.executable}")
data = fetch_california_housing()
X_train, X_test, y_train, y_test = train_test_split(data.data,
data.target, test_size=0.2, random_state=42)

with mlflow.start_run():
   mlflow.log_param("alpha", args.alpha)
   model = LinearRegression()
   model.fit(X_train, y_train)
   y_pred = model.predict(X_test)
   mse = mean_squared_error(y_test, y_pred)
   mlflow.log_metric("mse", mse)
   mlflow.sklearn.log_model(model, "model")
   print(f"MSE: {mse:.4f}")
```

In PyCharm's terminal, navigate to the project directory and run:

```
 mlflow run california_housing_project -P alpha=0.7
--env-manager=local
```

2. View the run in the MLFlow UI.

Questions:

1. What dependencies are listed in `requirements.txt`?
2. How did changing `alpha` affect the logged MSE?

Section 5: MLFlow Models

What is an MLFlow Model?
An MLFlow Model packages your trained model with metadata, like a ready-to-use app. It supports deployment on various platforms, ensuring consistency.

Key Features:

- **Logging Models**: Use `mlflow.sklearn.log_model()` to save.
- **Serving Models**: Deploy with `mlflow models serve`.
- **Cross-Platform**: Works with scikit-learn, PyTorch, etc.

How It Works:

1. Log a model during a run.
2. Serve it: `mlflow models serve -m runs:/<run_id>/model --port 1234`.
3. Send inference requests via HTTP.

Hands-on Exercise

Objective: Log and serve a linear regression model in PyCharm.

Code:

Model_serving.py (In your main directory)

```
import mlflow
import mlflow.sklearn
from sklearn.datasets import fetch_california_housing
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression

mlflow.set_tracking_uri("http://localhost:5000")

data = fetch_california_housing()
X_train, X_test, y_train, y_test = train_test_split(data.data,
data.target, test_size=0.2, random_state=42)

with mlflow.start_run(run_name="model_serving"):
```

```python
    model = LinearRegression()
    model.fit(X_train, y_train)
    mlflow.sklearn.log_model(model, "model")
    run_id = mlflow.active_run().info.run_id
    print(f"Model logged with run ID: {run_id}")
```

Run - In a new terminal: (Might take about 5 minutes to deploy)

Replace run_id with the run ID you got from previous step

```
mlflow models serve -m runs:/<run_id>/model --port 1234 --no-conda
```

If **facing errors** then run in terminal:

set MLFLOW_TRACKING_URI=http://localhost:5000

Then:

1. Open the Windows Start menu, search for “Environment Variables,” and select “Edit the system environment variables” or “Edit environment variables for your account.”
2. In the “System Properties” window, click “Environment Variables.”
3. Under “User variables” or “System variables,” click “New” or edit an existing MLFLOW_TRACKING_URI variable.
4. Set the variable name to MLFLOW_TRACKING_URI and the value to http://localhost:5000.
5. Click “OK” to save, then close all dialogs.

Restart Pycharm and then run this again after starting MLflow in new terminal:

```
mlflow models serve -m runs:/<run_id>/model --port 1234 --no-conda
```

Create inference.py

```python
import requests
import json

# Use the run ID from the model log
run_id = "run_id" #Replace with your run id
data = {"instances": [[-122.23, 37.88, 41.0, 880.0, 129.0,
322.0, 126.0, 8.3252]]}  # Example test data
try:
    response = requests.post("http://localhost:1234/invocations",
json=data, timeout=10)
```

```python
    response.raise_for_status()
    print("Prediction:", response.json())
except requests.exceptions.RequestException as e:
    print(f"Failed to connect to server: {e}")
```

Questions:

1. What metadata is included in the logged model?
2. How did the served prediction compare to local inference?
3. What are the benefits of serving with MLFlow?

Section 6: MLFlow Model Registry

Theory

What is the Model Registry?
The MLFlow Model Registry is a catalog for your models, tracking versions and stages (e.g., Staging, Production) with notes, perfect for team projects.

Key Features:

- **Register Models**: Use `mlflow.register_model`.
- **Manage Versions**: Transition stages with `mlflow.transition_model_version_stage`.
- **Annotations**: Add descriptions.

How It Works:

1. Register: `mlflow.register_model("runs:/<run_id>/model", "ModelName")`.
2. Update stages and descriptions.

Benefits:

- Centralizes model management.
- Tracks RL or vision model lifecycles.

Hands-on Exercise

Objective: Register and manage a model in the MLFlow Model Registry in PyCharm.

Code:

```python
import mlflow
from mlflow.tracking import MlflowClient

# Register model from a run
run_id = "<run_id_from_previous_run>"  # Replace with run ID
model_uri = f"runs:/{run_id}/model"
model_name = "California_Housing_Model"
mlflow.register_model(model_uri, model_name)
# Manage model versions
client = MlflowClient()
client.transition_model_version_stage(
    name=model_name,
    version=1,
    stage="Staging"
)
client.update_model_version(
    name=model_name,
    version=1,
    description="Linear regression model for California Housing"
)
```

Questions:

1. How does the Registry organize model versions?
2. What information was added to the description?
3. How can it help in team projects?