Day 12: MLflow Basics

Complete Guide to ML Lifecycle Management

Databricks 14-Days AI Challenge

MLflow - Tracking - Model Registry - Deployment

Agenda

- **Introduction to MLflow**
- **MLflow Components**
- **MLflow Tracking**
- **Model Registry**
- **MLflow Models**
- **Experiment Tracking**
- **Model Logging**
- **MLflow UI**
- **Practical Implementation**
- **Comparing Runs**
- **Best Practices**

What is MLflow?

Definition

MLflow is an **open-source platform** designed to manage the complete machine learning lifecycle.

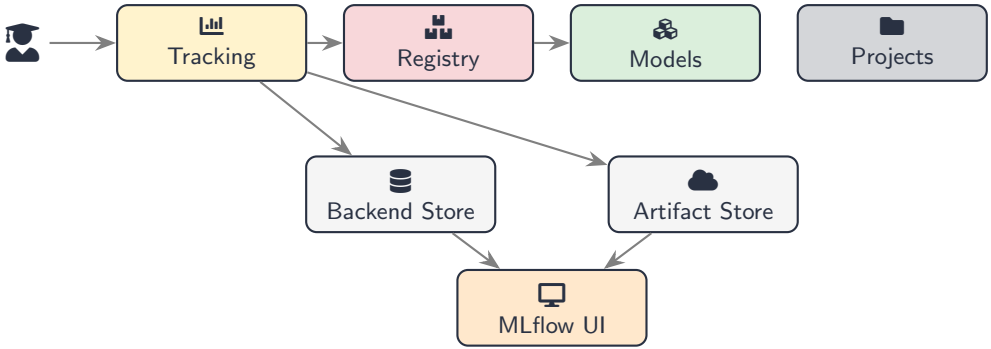
Key Challenges Solved:

- ▷ Reproducibility issues
- ▷ Experiment management
- ▷ Model versioning
- ▷ Deployment complexity
- ▷ Team collaboration

How MLflow Helps:

- ○ Tracks all parameters and versions
- ○ Centralized experiment history
- ○ Model Registry for versioning
- ○ Unified deployment format
- ○ Shared tracking server and UI

MLflow Architecture Overview



MLflow Components Overview

Component	Purpose	Key Features	When to Use
Tracking	Record and query experiments	Log params, metrics, artifacts	During model development
Registry	Centralized model store	Versioning, stage transitions	Managing production models
Models	Standard packaging format	Framework-agnostic, deploy options	Deploying to production
Projects	Reproducible runs	Package code with dependencies	Sharing code across envs

MLflow Tracking - Core Concepts

What is MLflow Tracking?

Component responsible for **logging and querying experiments**

What Gets Recorded:

- **Parameters:** Input configurations
- **Metrics:** Performance measurements
- **Artifacts:** Output files (models, plots)
- **Source:** Code version and entry point
- **Tags:** Custom metadata

Experiment Structure:

```
Experiment: "Customer Churn"
|-- Run 1: Random Forest
|          (accuracy: 0.85)
|-- Run 2: Logistic Regression
|          (accuracy: 0.78)
|-- Run 3: XGBoost
|          (accuracy: 0.89)
```

MLflow Tracking API

```python
import mlflow

# Experiment Management
mlflow.set_experiment("my_experiment")
mlflow.create_experiment("new_experiment")

# Run Management
mlflow.start_run(run_name="my_run")
mlflow.end_run()

# Logging Functions
mlflow.log_param("param_name", value)
mlflow.log_params({"p1": v1, "p2": v2})
mlflow.log_metric("metric_name", value)
mlflow.log_metrics({"m1": v1, "m2": v2})
mlflow.log_artifact("path/to/file")
mlflow.set_tag("tag_name", "value")
```

MLflow Model Registry

What is the Model Registry?

A **centralized model store** providing versioning, stage transitions, lineage tracking, and approval workflows.

Model Registry Operations

```python
from mlflow import MlflowClient

client = MlflowClient()

# Register a model
model_uri = "runs:/<run_id>/model"
mlflow.register_model(model_uri, "MyModelName")

# Transition model stage
client.transition_model_version_stage(
    name="MyModelName",
    version=1,
    stage="Production"
)

# Add descriptions
client.update_registered_model(
    name="MyModelName",
    description="Customer churn prediction using XGBoost"
)
```

MLflow Models - Standard Packaging

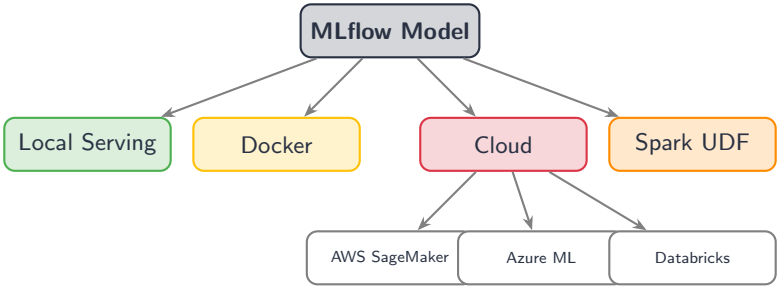
Model Flavors:

- `mlflow.sklearn` - Scikit-learn
- `mlflow.pytorch` - PyTorch
- `mlflow.tensorflow` - TensorFlow
- `mlflow.xgboost` - XGBoost
- `mlflow.spark` - Spark MLlib
- `mlflow.pyfunc` - Custom models
- `mlflow.transformers` - Hugging Face

Model Directory Structure:

```
model/
|-- MLmodel              # Metadata
|-- conda.yaml           # Conda env
|-- requirements.txt     # Dependencies
|-- python_env.yaml      # Python env
|-- model.pkl            # Serialized model
|-- input_example.json
```

Model Deployment Options



Model URI Formats

URI Format	Description	Example
`runs:/<run_id>/<path>`	Load from specific run	`runs:/a1b2c3d4/model`
`models:/<name>/<version>`	Load specific version	`models:/MyModel/3`
`models:/<name>/<stage>`	Load from stage	`models:/MyModel/Production`
`models:/<name>/latest`	Load latest version	`models:/MyModel/latest`

Experiment Tracking Setup

Three Ways to Configure Tracking:

Local Tracking

```
import mlflow

# Logs to ./mlruns
mlflow.set_tracking_uri(
    "mlruns"
)
```

Remote Server

```
import mlflow

# Logs to remote
mlflow.set_tracking_uri(
    "http://server:5000"
)
```

Databricks

```
import mlflow

# Logs to Databricks
mlflow.set_tracking_uri(
    "databricks"
)
```

Parameters vs Metrics vs Artifacts

Aspect	Parameters	Metrics	Artifacts
What	Configuration values	Performance measurements	Files and objects
When Set	Before/during training	During/after training	After computation
Data Type	String (converted)	Numeric	Any file
Example	`learning_rate=0.01`	`accuracy=0.95`	`model.pkl`
Searchable	Yes	Yes	No (metadata only)

Model Logging - Complete Example

```python
from mlflow.models.signature import infer_signature

with mlflow.start_run(run_name="complete_example"):
    # Train model
    model = LinearRegression()
    model.fit(X_train, y_train)

    # Create signature
    signature = infer_signature(X_train, model.predict(X_train))

    # Log model with all options
    mlflow.sklearn.log_model(
        sk_model=model,
        artifact_path="model",
        signature=signature,
        input_example=X_train.iloc[:3],
        registered_model_name="PurchasePredictionModel",
        pip_requirements=["scikit-learn==1.0.2"],
        metadata={"author": "data-team"}
    )
```

Loading Logged Models

```
# Method 1: Load from run
model_uri = f"runs:/{run_id}/model"
loaded_model = mlflow.sklearn.load_model(model_uri)

# Method 2: Load from registry
model_uri = "models:/PurchasePredictionModel/Production"
loaded_model = mlflow.sklearn.load_model(model_uri)

# Method 3: Load as generic Python function
loaded_model = mlflow.pyfunc.load_model(model_uri)
predictions = loaded_model.predict(X_test)
```

MLflow UI

Starting the UI:

```
# Default port 5000
mlflow ui

# Custom port
mlflow ui --port 8080

# With backend store
mlflow ui --backend-store-uri
    sqlite:///mlflow.db
```

Key UI Features:

- ▷ Runs Table - Filter and sort
- ▷ Run Details - Debug access
- ▷ Compare Runs - Side-by-side
- ▷ Chart View - Visualizations
- ▷ Search - SQL-like filtering
- ▷ Download - Export as CSV

MLflow Search Syntax

```
# Search by metrics
mlflow.search_runs(
    experiment_ids=["1"],
    filter_string="metrics.r2_score > 0.8"
)

# Search by parameters
mlflow.search_runs(
    filter_string="params.model_type = 'LinearRegression'"
)

# Combined search
mlflow.search_runs(
    filter_string="metrics.rmse < 100 AND params.epochs > '50'"
)

# Search by tags
mlflow.search_runs(
    filter_string="tags.team = 'data-science'"
)
```

Practical Implementation - Data Preparation

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

# Prepare data from Delta Lake
df = spark.table("gold.products").toPandas()
X = df[["views", "cart_adds"]]
y = df["purchases"]
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.2, random_state=42
)
```

Understanding the Data

Features: `views` (page views), `cart_adds` (cart additions)
Target: `purchases` - what we want to predict

Complete MLflow Experiment

```
mlflow.set_experiment("Purchase Prediction")

with mlflow.start_run(run_name="linear_regression_v1"):
    # Log parameters
    mlflow.log_param("model_type", "LinearRegression")
    mlflow.log_param("test_size", 0.2)
    mlflow.set_tag("team", "data-science")

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

    # Evaluate and log metrics
    r2_score = model.score(X_test, y_test)
    mlflow.log_metric("r2_score", r2_score)

    # Log model with signature
    signature = infer_signature(X_train, model.predict(X_train))
    mlflow.sklearn.log_model(model, "model", signature=signature)

    print(f"R2 Score: {r2_score:.4f}")
```

Understanding R-Squared Score

R-Squared (Coefficient of Determination)

$$R^2 = 1 - \frac{SS_{res}}{SS_{tot}} = 1 - \frac{\sum_{i=1}^{n}(y_i - \hat{y}_i)^2}{\sum_{i=1}^{n}(y_i - \bar{y})^2}$$

R-Squared Value	Interpretation
1.0	Perfect prediction
0.8 - 1.0	Excellent fit
0.6 - 0.8	Good fit
0.4 - 0.6	Moderate fit
0.0 - 0.4	Poor fit
Less than 0	Worse than mean prediction

Comparing Multiple Models

```python
from sklearn.linear_model import LinearRegression, Ridge, Lasso
from sklearn.ensemble import RandomForestRegressor

models = {
    "LinearRegression": LinearRegression(),
    "Ridge": Ridge(alpha=1.0),
    "Lasso": Lasso(alpha=0.1),
    "RandomForest": RandomForestRegressor(n_estimators=100)
}

for model_name, model in models.items():
    with mlflow.start_run(run_name=model_name):
        mlflow.log_param("model_type", model_name)
        model.fit(X_train, y_train)
        score = model.score(X_test, y_test)
        mlflow.log_metric("r2_score", score)
        mlflow.sklearn.log_model(model, "model")
        print(f"{model_name}: R2 = {score:.4f}")
```

Model Comparison Results

Model	R-Squared	RMSE	MAE	Status
RandomForest	**0.8923**	**45.23**	**32.15**	
Ridge	0.8156	58.92	42.87	
LinearRegression	0.8134	59.34	43.21	
Lasso	0.7892	63.12	46.54	

Next Steps

- ✔ Register winning model to Model Registry
- ✔ Promote to Staging for validation
- ✔ Deploy to Production after approval

Best Practices - Experiment Organization

Practice	Description	Example
Meaningful Names	Use descriptive experiment names	`purchase_prediction_q4`
Consistent Naming	Follow naming conventions	`{task}_{model}_{ver}`
Tags for Filtering	Add searchable metadata	`team`, `dataset`, `purpose`
Run Names	Make runs identifiable	`rf_n100_depth10`

Best Practices - What to Log

✔ Always Log:

- All hyperparameters
- Train and test metrics
- Model artifacts
- Data version/hash
- Code version (git commit)

💡 Consider Logging:

- Feature importance
- Confusion matrices
- Learning curves
- Sample predictions
- Data statistics

Common Mistakes to Avoid

Mistake	Problem	Solution
Not setting experiment	Runs go to default	Always call `set_experiment()`
Missing parameters	Cannot reproduce results	Log ALL configuration
No signatures	Deployment issues	Always include model signature
Large artifacts	Slow tracking	Use artifact stores for large files
Forgetting `end_run()`	Orphaned runs	Use context manager (`with`)

Recommended Code Structure

```python
def train_model(params):
    """Train and log model with MLflow."""
    with mlflow.start_run(run_name=params.get("run_name")):
        # 1. Log parameters first
        mlflow.log_params(params)

        # 2. Train model
        model = create_model(params)
        model.fit(X_train, y_train)

        # 3. Evaluate
        metrics = evaluate_model(model, X_test, y_test)

        # 4. Log metrics
        mlflow.log_metrics(metrics)

        # 5. Log artifacts (visualizations)
        log_visualizations(model, X_test, y_test)

        # 6. Log model last
        mlflow.sklearn.log_model(model, "model")

        return model, metrics
```

Quick Reference Card

Task	Code
Set experiment	`mlflow.set_experiment("name")`
Start run	`with mlflow.start_run():`
Log parameter	`mlflow.log_param("key", value)`
Log metric	`mlflow.log_metric("key", value)`
Log model	`mlflow.sklearn.log_model(model, "path")`
Load model	`mlflow.sklearn.load_model("uri")`
Register model	`mlflow.register_model(uri, "name")`
Start UI	`mlflow ui --port 5000`

Summary - Key Takeaways

Components

- Tracking
- Model Registry
- Models
- Projects

Workflow

1. Set Experiment
2. Start Run
3. Log Parameters
4. Train Model
5. Log Metrics
6. Log Model
7. Compare and Register

Best Practices

- Meaningful names
- Complete logging
- Model signatures
- Version control
- Use context managers

Next Steps

After Mastering MLflow Basics, Explore:

1. **MLflow Projects:** Package code for reproducible runs
2. **Model Serving:** Deploy models as REST APIs
3. **AutoML Integration:** Combine with tools like Hyperopt
4. **Custom Flavors:** Create flavors for custom frameworks
5. **MLflow Recipes:** Standardized ML workflows

Resources:

MLflow Documentation — MLflow on Databricks

Thank You!

Day 12: MLflow Basics Complete

Easy AI Labs Yash Kavaiya Gen AI Guru