MLFlow

Machine Learning Workflow, Tracking, Deployment Framework

MLFlow Overview

Manages Lifecycle

- Experiment tracking
  - Parameters
  - Metrics
  - Artifacts
- Reproducibility
  - Package Resources based on captured run
- Deployment
  - Deploy to different platforms
    - Local service
    - Docker Image
    - Cloud

MLFlow getting started

Python

```
pip install mlflow
```

R

```r
install.packages("mlflow")
```

```r
mlflow::install_mlflow()
```

Experimentation - MLFlow Tracking

Python

```python
import os
from mlflow import log_metric, log_param, log_artifact
if __name__ == "__main__":
    # Log a parameter (key-value pair)
    log_param("param1", 5)
    # Log a metric; metrics can be updated throughout
the run
    log_metric("foo", 1)
    log_metric("foo", 2)
    log_metric("foo", 3)
    # Log an artifact (output file)
    with open("output.txt", "w") as f:
        f.write("Hello world!")
    log_artifact("output.txt")
```

R

```r
library(mlflow)

# Log a parameter (key-value pair)
mlflow_log_param("param1", 5)

# Log a metric; metrics can be updated throughout the
run
mlflow_log_metric("foo", 1)
mlflow_log_metric("foo", 2)
mlflow_log_metric("foo", 3)

# Log an artifact (output file)
writeLines("Hello world!", "output.txt")
mlflow_log_artifact("output.txt")
```

MLFlow Tracking UI

Python

miflow ui

http://localhost:5000

R

```
mlflow_ui()
```

http://localhost:5000

MLFlow Tracking Server

Launch Tracking Server

Backend Stores

`--backend-store-uri`

- `local file path`
- `database server`

Artifact Stores

`--default-artifact-root`

- Amazon S3
- Azure Blob Storage
- Google Cloud Storage
- FTP server
- SFTP Server
- NFS
- HDFS

Running Server:

```
mlflow server \
    --backend-store-uri
/mnt/persistent-disk \
    --default-artifact-root
s3://my-mlflow-bucket/ \
    --host 0.0.0.0
```

Using the Server:

- export Environment variable **`MLFLOW_TRACKING_URI`**
- set url in code `mlflow_set_tracking_uri(remote_server_uri)`

Python

```python
import mlflow
remote_server_uri = "..." # set to your server URI
mlflow.set_tracking_uri(remote_server_uri)
# Note: on Databricks, the experiment name passed to mlflow_set_experiment must be a
# valid path in the workspace
mlflow.set_experiment("/my-experiment")
with mlflow.start_run():
    mlflow.log_param("a", 1)
        mlflow.log_metric("b", 2)
```

R

```r
library(mlflow)
install_mlflow()
remote_server_uri = "..." # set to your server URI
mlflow_set_tracking_uri(remote_server_uri)
# Note: on Databricks, the experiment name passed to mlflow_set_experiment must be a
# valid path in the workspace
mlflow_set_experiment("/my-experiment")
mlflow_log_param("a", "1")
```

Packaging ML Code - MLFlow Projects

- MLFlowProject YAML Configuration File
- or Convention
- Environments
  - Conda
  - Docker Container
  - System (running local environment)
- Project Directories
  - MLproject file
  - environment file:
    - conda_env: environment.yaml
    - docker_env: environment.yaml
    - or run with --no-conda to use system env.

MLproject

```
name: My Project
conda_env: environment.yaml
# Can have a docker_env instead of a conda_env, e.g.
# docker_env:
#    image:  mlflow-docker-example
entry_points:
  main:
    parameters:
      data_file: path
      regularization: {type: float, default: 0.1}
    command: "python train.py -r {regularization} {data_file}"
  validate:
    parameters:
      data_file: path
    command: "python validate.py {data_file}"
```

MLFlow Projects

Commands

- entry point names
- parameters
  - Name, Type and Default
  - Type can be String, float, path, uri
- command
  - Bash command following Python Format syntax

```
entry_points:
  main:
    parameters:
      data_file: path
      regularization: {type: float, default: 0.1}
    command: "python train.py -r {regularization}
{data_file}"
  validate:
    parameters:
      data_file: path
    command: "python validate.py {data_file}"
```

MLFlow Project - Running

`mlflow run` Command line tool

or

mlflow.projects.run() API call in python code.

Target environments:

- Databricks on AWS or Azure
- Kubernetes cluster
- run synchronously locally

Package Built Models - MLFlow Models

Supports multiple flavors of ML Frameworks and Platforms:

- Python Function (python_function)
- R Function (crate)
- H_2O (h2o)
- Keras (keras)
- MLeap (mleap)
- PyTorch (pytorch)
- Scikit-learn (sklearn)
- Spark MLlib (spark)
- TensorFlow (tensorflow)
- ONNX (onnx)

Package and deploy models to target platforms:

- Amazon SageMaker
- Apache Spark UDF
- Microsoft Azure ML

Package models as Docker containers

`mlflow models build-docker`

Predict

`mlflow models predict`

Serve

`mlflow models serve`

MLFlow Examples

quickstart

docker

flower classifier

hyperparam

sklearn_diabetes

tensorflow

Useful Links

mlflow.org

mlflow docs

mlflow code in github

mlflow community

mlflow community on Slack

mlflow stack overflow

Built-in Integrations

Built-in integrations:

TensorFlow
PyTorch
K Keras
APACHE Spark
scikit learn
H_2O.ai

python
R
Java
mleap
CONDA
kubernetes

docker
Amazon SageMaker
Azure Machine Learning
Google Cloud

Advantages

- Tracking training runs
  - Capture parameters
  - Capture metrics
  - Store Artifacts including
    - Generated Images
    - Models
    - Preprocessing Metrics
  - Compare Runs
- Packaging Models
  - Build Docker Images
  - Create Archives
  - Push to Github
- Deployment
  - Run locally
  - Deploy to AWS
  - Deploy to Azure
  - Deploy to Databricks
  - Deploy to Kubernetes

Disadvantages

- New Framework / Platform
- Growing Community but not mature yet
- More complex scenarios are non-intuitive to configure / use
- Tricky to debug those more complex scenarios

Outlook

- Expected to continue to be supported
- Community is slowly growing
- Needs better integration with Tensorflow / Pytorch and others
- Needs smoother configuration of non-standalone models scenarios