



Módulo 4: MLFlow Projects

Introduction to MLFlow Projects

MLFlow Projects simplifica el ciclo de vida del ML proporcionando una manera de organizar y ejecutar código de ML en una forma reproducible. Incluye código usado para entrenar y construir modelos, seguimiento de experimentos y registro de modelos en Model Registry. Projects se usan para empaquetar el código en unidades reusables que permitan una colaboración simple entre usuarios. Proporcionan portabilidad para correr el código en diferentes ambientes como máquinas locales y la nube.

Un Project es un directorio de archivos que contienen nuestro código de ML. Puede guardarse en un repositorio de Git. Usa un archivo llamado MLproject para describir el proyecto. Este es un archivo YAML, que es un formato legible para humanos, comúnmente usado para archivos de configuración.

MLproject file

- `name:` - Name of the Project
- `entry_points:`
 - Command(s) to run
 - `.py` and `.sh` files
 - Workflows
- `python_env:`
 - Python environment
 - `conda.yaml` or `python_env.yaml`

MLproject example

```
name: salary_model
entry_points:
  main:
    command: "python train_model.py"
python_env: python_env.yaml
```

train_model.py

```
# Import libraries and modules
import mlflow
import mlflow.sklearn
import pandas as pd
from sklearn.linear_model import LinearRegression
from sklearn.model_selection import train_test_split

# Training Data
df = pd.read_csv('Salary_predict.csv')
X = df[["experience", "age", "interview_score"]]
y = df[["Salary"]]
X_train, X_test, y_train, y_test = train_test_split(X, y, train_size=0.7, random_state=0)
```

```
# Set Auto logging for Scikit-learn flavor
mlflow.sklearn.autolog()

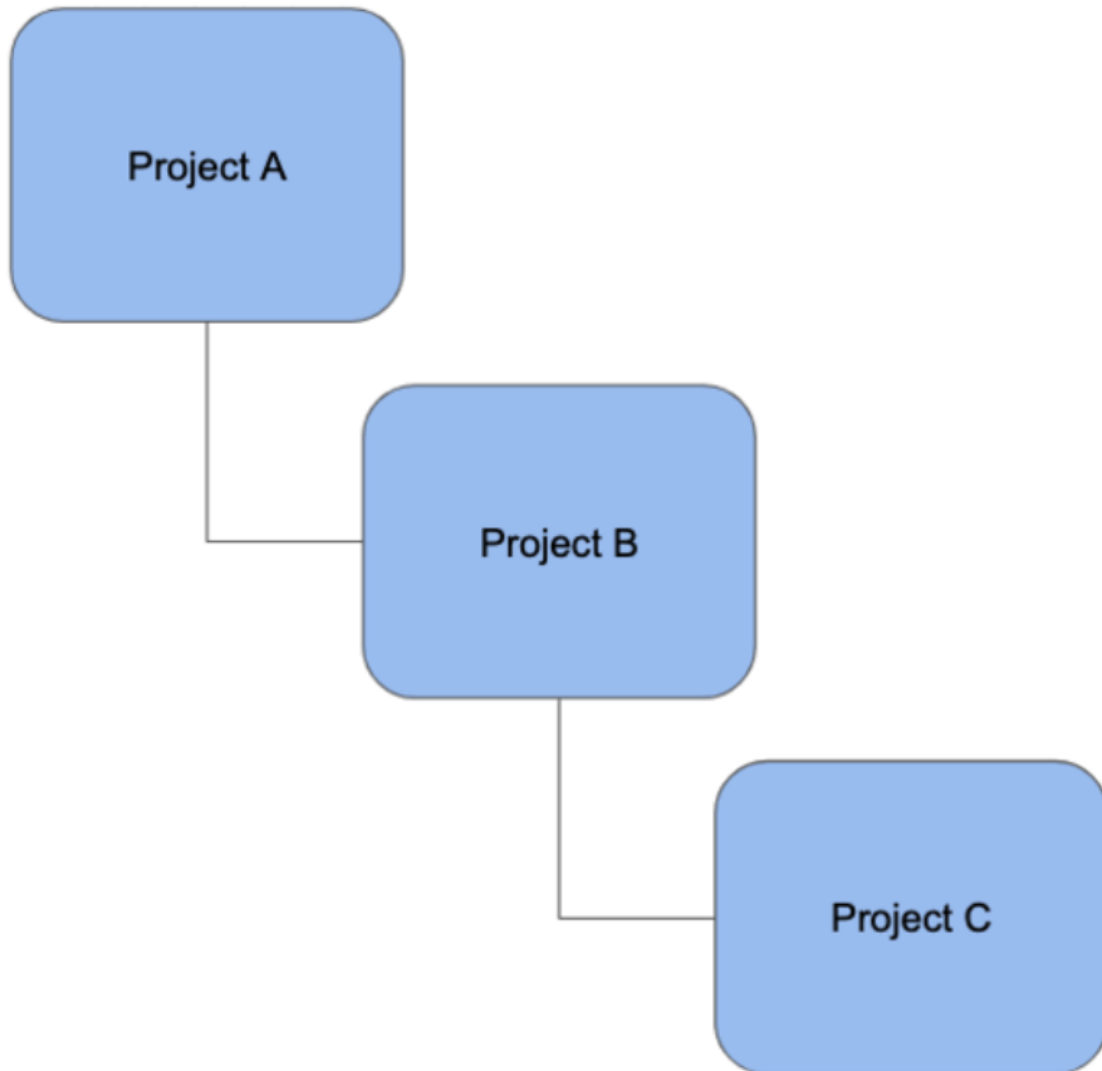
# Train the model
lr = LinearRegression()
lr.fit(X_train, y_train)
```

python_env.yaml

```
python: 3.10.8
build_dependencies:
- pip
- setuptools
- wheel
dependencies:
- -r requirements.txt
```

Running MLFlow Projects

MLFlow Projects pueden ejecutarse de manera programada utilizando la API de MLFlow o también a través de la línea de comandos.



Projects API

```
mlflow.projects
```

```
mlflow.projects.run()
```

- `uri` - URI to MLproject file
- `entry_point` - Entry point to start run from MLproject
- `experiment_name` - Experiment to track training run
- `env_manager` - Python environment manager: `local`, `virtualenv`, or `conda`

```
# Run MLflow Project
mlflow.projects.run(
    uri='./',
    entry_point='main',
    experiment_name='My Experiment',
    env_manager='conda'
)
```

Projects run

```
# Import MLflow Module
```

```
import mlflow
```

```
# Run local Project
```

```
mlflow.projects.run(uri='./', entry_point='main',
                    experiment_name='Salary Model')
```

`./` representa el mismo directorio en el que el código es ejecutado.

```
2023/04/02 14:33:23 INFO mlflow.projects: 'Salary Model' does not exist.
Creating a new experiment
2023/04/02 14:33:23 INFO mlflow.utils.virtualenv: Installing python 3.10.8 if it
does not exist
2023/04/02 14:33:23 INFO mlflow.utils.virtualenv: Creating a new environment
./mlflow/envs/mlflow-44f5094bba686a8d4a5c772
created virtual environment CPython3.10.8.final.0-64 in 236ms
2023/04/02 14:33:23 INFO mlflow.utils.virtualenv: Installing dependencies
```

```
2023/04/02 14:33:59 INFO mlflow.projects.backend.local: === Running command
'source /.mlflow/envs/mlflow-44f5094bba686a8d4a5c772/bin/activate && python
train_model.py' in run with ID '562916d45aeb48ec84c1c393d6e3f5b6' ===
2023/04/02 14:34:34 INFO mlflow.projects: === Run (ID
'562916d45aeb48ec84c1c393d6e3f5b6') succeeded ===
```

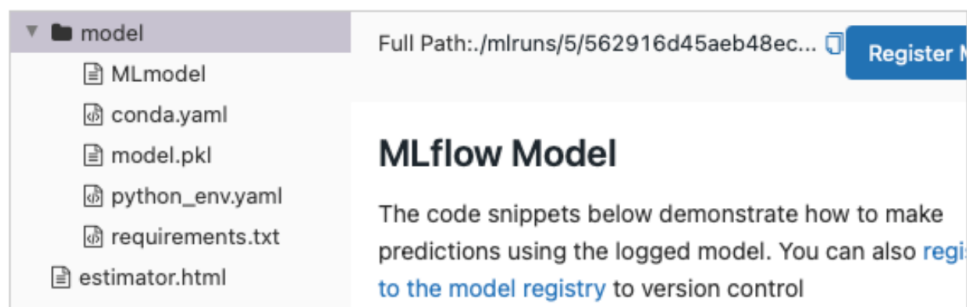
MLflow Tracking

> Parameters (4)

> Metrics (5)

> Tags (2)

✓ Artifacts



Full Path:../mlruns/5/562916d45aeb48ec... [Register Model](#)

MLflow Model

The code snippets below demonstrate how to make predictions using the logged model. You can also [register the model to the model registry](#) to version control

Command line

Command line

```
mlflow run
```

- `--entry-point` - Entry point to start run from MLproject
- `--experiment-name` - Experiment to track training run
- `--env-manager` - Python environment manager: `local`, `virtualenv`, or `conda`
- `URI` - URI to MLproject file

```
# Run main entry point from Salary Model experiment
mlflow run --entry-point main --experiment-name "Salary Model" ./
```

```
2023/04/02 15:23:34 INFO mlflow.utils.virtualenv: Installing python 3.10.8 if it
does not exist
2023/04/02 15:23:34 INFO mlflow.utils.virtualenv: Environment
/.mlflow/envs/mlflow-44f5094bba686a8d4a5c772 already exists
2023/04/02 15:23:34 INFO mlflow.projects.backend.local: === Running command 'source
/Users/weston/.mlflow/envs/mlflow-44f5094bba686a8d4a5c772/bin/activate && python
train_model.py' in run with ID 'da5b37b6f53245e5bca59ba8ed6d7dc1' ===
2023/04/02 15:23:38 INFO mlflow.projects: === Run
(ID 'da5b37b6f53245e5bca59ba8ed6d7dc1') succeeded ===
```

Specifying parameters

MLFlow Projects permiten la flexibilidad y customización a través del uso de parámetros. Los parámetros son variables que el usuario puede especificar en la ejecución del MLFlow Project. Usar parámetros simplifica el proceso de probar diferentes configuraciones para los modelos de ML, como los hiperparámetros.

```
parameter_1_name:
  type: data_type
  default: default_value
parameter_2_name:
  type: data_type
  default: default_value
```

Data types

- float, string, etc...
- defaults to string

Default value

- parameter value if nothing specified during run
- True, False, None, etc...

Parameters block

```
name: project_name
conda_env: python_env.yaml
entry_points:
  main:
    parameters:
      parameter_1:
        type: data_type
        default: default_value
      parameter_2:
        type: data_type
        default: default_value
    command: "python train.py {parameter_1_name} {parameter_2_name}"
```

```
# Import libraries and modules
```

```
import mlflow
```

```
import mlflow.sklearn
```

```
import pandas as pd
```

```
from sklearn.linear_model import LinearRegression
```

```
from sklearn.model_selection import train_test_split
```

```
# Training Data
```

```
df = pd.read_csv('Salary_predict.csv')
```

```
X = df[["experience", "age", "interview_score"]]
```

```
y = df[["Salary"]]
```

```

# Train test split
X_train, X_test, y_train, y_test = train_test_split(X, y, train_size=0.7,
                                                    random_state=0)

# Set Auto logging for Scikit-learn flavor
mlflow.sklearn.autolog()

# Parameters
n_jobs_param = int(sys.argv[1])
fit_intercept_param = bool(sys.argv[2])

# Train model
lr = LinearRegression(n_jobs=n_jobs_param, fit_intercept=fit_intercept_param)
lr.fit(X_train, y_train)

```

MLproject

```

name: salary_model
python_env: python_env.yaml
entry_points:
  main:
    parameters:
      n_jobs_param:
        type: int
        default: 1
      fit_intercept_param:
        type: bool
        default: True
    command: "python train_model.py {n_jobs_param} {fit_intercept_param}"

```

Running parameters

Python

```
mlflow.projects.run()
```

```

parameters={
    'parameter_1': data_value,
    'parameter_2': data_value,
}

```

CLI

```
mlflow run
```

```
-P parameter_1=parameter_1_value
```

```
-P parameter_2=parameter_2_value
```

Projects run

```
# Import MLflow Module
import mlflow

# Run local Project
mlflow.projects.run(
    uri='./', entry_point='main',
    experiment_name='Salary Model',
    parameters={
        'n_jobs_param': 2,
        'fit_intercept_param': False
    })
```

Run command

```
# Run main entry point from Salary Model experiment
mlflow run --entry-point main --experiment-name "Salary Model" \
    -P n_jobs_param=3 -P fit_intercept_param=True ./
```

Workflows

MLFlow Projects permite ejecutar flujos de trabajo de múltiples pasos.

MLproject

```
name: project_name
python_env: python_env.yaml
entry_points:
  step_1:
    command: "python train_model.py"
  step_2:
    command: "python evaluate_model.py {run_id}"
parameters:
  run_id:
    type: str
    default: None
```

Workflows

```
import mlflow

# Step 1
step_1 = mlflow.projects.run(
    uri='./',
    entry_point='step_1'
)

# Step 2
step_2 = mlflow.projects.run(
    uri='./',
    entry_point='step_2'
)
```

Cada vez que llamamos al método de projects.run este nos regresa un run object. Los atributos de este objeto pueden pasarse a otros pasos en el flujo de trabajo como parámetros.

`step_1.cancel()` - Terminate a current run

`step_1.get_status()` - Get the status of a run

`step_1.run_id` - `run_id` of the run

`step_1.wait()` - Wait for the run to finish

Projects run

```
import mlflow
```

```
# Step 1
```

```
step_1 = mlflow.projects.run(  
    uri='./',  
    entry_point='step_1'  
)
```

```
# Set variable for step_1 run_id
```

```
step_1_run_id = step_1.run_id
```

```
# Step 2
```

```
step_2 = mlflow.projects.run(  
    uri='./',  
    entry_point='step_2',  
    parameters={  
        'run_id': step_1_run_id  
    }  
)
```