Introduction to MLflow

1. Introduction to MLflow

What is MLflow?

- MLflow
  - 머신 러닝 라이프사이클의 어렵지만 필요한 단계를 단순화하고 자동화하려고 시도함
  - 실험 추적, 재현성, 배포 및 중앙 집중식 모델 레지스트리를 제공하여 사용자가 ML 라이프사이클을 관리하는 데 도움이 되는 오픈 소스 플랫폼
  - 구성 요소
    - MLflow Tracking: 모델 실험에서 데이터를 기록하고 쿼리하는 데 사용, 필요할 때 쉽게 검색할 수 있는 모델, 코드 및 기타 아티팩트를 저장하는 데 사용
    - MLflow Models: 사용자가 모델을 패키징하는 방식을 표준화하여 보다 간소화된 배포 프로세스를 사용 가능, 모델 로딩 및 모델 추론과 같은 사용자 지정을 빌드하는 방법도 제공
    - Model Registry: 특정 개발 환경에 대해 버전 제어 및 태그가 지정될 수 있는 모델에 대한 중앙 집중식 저장소를 제공
    - MLflow Projects: 다양한 환경, 플랫폼 및 사용자 작업 공간에서 재생산할 수 있도록 ML 코드 패키징을 표준화하는 데 사용, 프로젝트 또한 자동화를 지원하기 위해 반복성을 제공

```
# Import MLflow
import mlflow

# Create new experiment
mlflow.create_experiment("Unicorn Model")

# Tag new experiment
mlflow.set_experiment_tag("version", "1.0")

# Set the experiment
mlflow.set_experiment("Unicorn Model")
```

MLflow Tracking

- 사용자가 API 를 통해 메트릭과 매개변수를 추적할 수 있도록 하는 MLflow의 구성 요소
- 사용자가 코드나 다른 파일 유형과 같은 artifacts를 저장할 수 있도록 한다.
- MLflow는 데이터난 artifacts가 MLflow Tracking 에 저장될 때 “로깅” 이라는 용어를 사용한다.
- “runs” 라는 개념을 중심으로 구성됨 → 새로운 run 은 새로운 모델 교육을 의미하며 모델에 대한 정보가 기록된다.
    - 각 run 은 실험 내에 배치된다.
- logging

Logging to MLflow Tracking

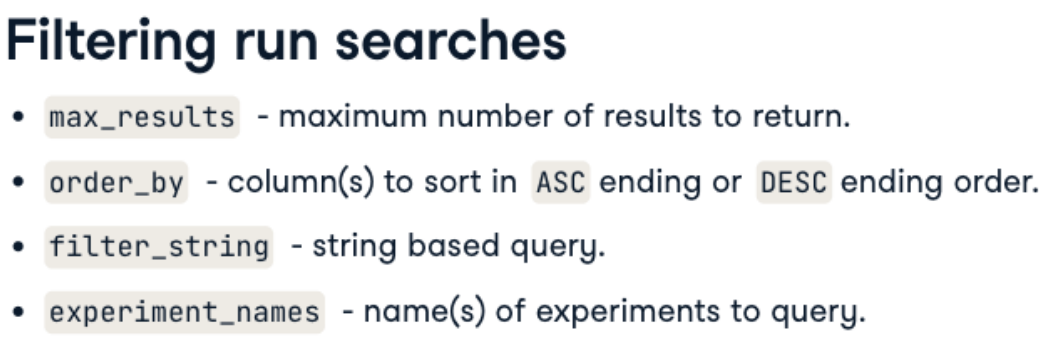
- **Metrics**
    - `log_metric("accuracy", 0.90)`
    - `log_metrics({"accuracy": 0.90, "loss": 0.50})`
- **Parameters**
    - `log_param("n_jobs", 1)`
    - `log_params({"n_jobs": 1, "fit_intercept": False})`
- **Artifacts**
    - `log_artifact("file.py")`
    - `log_artifacts("./directory/")`

```
# Set the experiment
mlflow.set_experiment("Unicorn Sklearn Experiment")

# Start a run
mlflow.start_run()
```

```
# Log metrics
mlflow.log_metric("r2_score", r2_score)

# Log parameter
mlflow.log_param("n_jobs", n_jobs)

# Log the training code
mlflow.log_artifact("train.py")
```

Querying runs

- 메트릭과 다른 데이터를 비교하여 ML 애플리케이션에서 어떤 모델을 사용할 지 결정해야 함
- 실행 정보를 수집하는 방법
  - `search_runs` 함수
    - 유연하며, 필요에 맞게 데이터를 검색하기 위해 여러 가지 다른 인수를 사용할 수 있다.



  - 쿼리의 결과에 따라 모델 결정하게 됨

```
# Create a filter string for R-squared score
r_squared_filter = "metrics.r2_score > .70"

# Search runs
mlflow.search_runs(experiment_names=["Unicorn Sklearn Ex
periments", "Unicorn Other Experiments"],
```

```
                      filter_string=r_squared_filter,
                      order_by=["metrics.r2_score DESC"])
```

2. MLflow Models

Indroduction to MLflow Models

- MLflow 모델의 구성 요소를 사용하여 머신 러닝 모델을 패키징하는 표준화된 방법을 제공
- MLflow 모델
  - 모델을 표준화하면 인기 있는 ML 라이브러리와 배포 도구 간의 쉬운 통합이 가능하다.
  - Flavors 라는 강력한 개념을 사용한다.
    - 모델의 로킹, 패키징 및 로디 프로세스를 간소화하여 사용자 지정 코드를 작설할 필요성을 최소화한다.
    - Flavor-dot-autolog 를 사용하여 “자동 로킹” 이라는 메서드를 지원한다.
    - 저장 형식 → yaml

```
# Import Scikit-learn flavor
import mlflow.sklearn

# Set the experiment to "Sklearn Model"
mlflow.set_experiment("Sklearn Model")

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

lr = LinearRegression()
lr.fit(X_train, y_train)

# Get a prediction from test data
print(lr.predict(X_test.iloc[[5]]))
```

Model API

Mlflow 가 Model API 를 사용하여 모델을 저장하고 로드하는 방법

- MLflow 는 사용자가 MLflow 의 모든 구성 요소에서 정보를 프로그래밍 방식으로 생성, 나열 및 검색할 수 있는 REST API 를 사용한다.
- 모델 API → 로드 모델, 모델 저장, 모델 로드

```
# Load model from local filesystem
model = mlflow.sklearn.load_model("lr_local_v1")

# Training Data
X = df[["R&D Spend", "Administration", "Marketing Spend",
"State"]]
y = df[["Profit"]]
X_train, X_test, y_train, y_test = train_test_split(X, y, t
rain_size=0.7,random_state=0)
# Train Model
model.fit(X_train, y_train)

# Save model to local filesystem
mlflow.sklearn.save_model(model, "lr_local_v2")


# Log model to MLflow Tracking
mlflow.sklearn.log_model(lr_model, "lr_tracking")

# Get the last run
run = mlflow.last_active_run()

# Get the run_id of the above run
run_id = run.info.run_id

# Load model from MLflow Tracking run_id and path used to l
og model
model = mlflow.sklearn.load_model(f"runs:/{run_id}/lr_track
ing")
```

Custom models

모든 머신 러닝 애플리케이션은 다르고 요구 사항도 다르다. → 사용자 정의 모델 클래스를 만든다.

Custom model class

- Custom model class
  - `MyClass(mlflow.pyfunc.PythonModel)`
- PythonModel class
  - `load_context()` - loads artifacts when `mlflow.pyfunc.load_model()` is called
  - `predict()` - takes model input and performs user defined evaluation

```
# Create Python Class
class CustomPredict(mlflow.pyfunc.PythonModel):
    # Method for loading model
    def load_context(self, context):
        self.model = mlflow.sklearn.load_model("./lr_mode
l/")
    # Method for custom inference
    def predict(self, context, model_input):
        predictions = self.model.predict(model_input)
        decoded_predictions = []
        for prediction in predictions:
            if prediction == 0:
                decoded_predictions.append("female")
            else:
                decoded_predictions.append("male")
        return decoded_predictions


# Log the pyfunc model
mlflow.pyfunc.log_model(
    artifact_path="lr_pyfunc",
    # Set model to use CustomPredict Class
    python_model=CustomPredict(),
    artifacts={"lr_model": "lr_model"}
)
```

```
run = mlflow.last_active_run()
run_id = run.info.run_id

# Load the model in python_function format
loaded_model = mlflow.pyfunc.load_model(f"runs:/{run_id}/lr
_pyfunc")


# Eval Data
eval_data = X_test
eval_data["sex"] = y_test
# Log the model using Scikit-Learn Flavor
mlflow.sklearn.log_model(lr_class, "model")

# Get run id
run = mlflow.last_active_run()
run_id = run.info.run_id

# Evaluate the logged model with eval_data data
mlflow.evaluate(f"runs:/{run_id}/model",
                data=eval_data,
                targets="sex",
                model_type="classifier"
)
```

Model serving

모델 배포를 MLflow tkdyd

- REST API
  - MLflow 는 모델을 REST API 로 제공한다.
  - 모델을 배포하는 데 사용되는 MLflow 의 API 는 네 가지 엔드포인트를 정의한다.
    - ping, health 엔드 포인트 : 서비스에 대한 상태 정보를 가져오는 데 사용
    - version 엔드 포인트 : MLflow 의 버전을 검색하는 데 사용
    - invocations 엔드 포인트 : 배포된 모델에서 점수를 검색하는 데 사용

→ 모델이 배포되면 MLflow 가 실행되는 URL 로 이동하여 각 엔드포인트에 도달할 수 있다.

- 호출 엔드포인트
  - CSV 또는 JSON 을 입력으로 허용한다.

3. MLflow Model Registry

Introduction to MLflow Model Registry

- 모델 수명주기
  - 개발, 스테이징, 프로덕션과 같은 다양한 소프트웨어 환경을 통해 모델의 라이프사이클을 관리하는 것이 중요하다.
- MLflow Model Registry
  - UI 와 MLflow Client 모듈을 통해 협업을 위한 모델에 대한 엑세스를 제공한다.
    - MLflow 클라이언트 모듈은 실험, 실행, 모델 버전 및 등록된 모델과 상호 작용하는 프로그래밍 방식을 제공한다. 클라이언트 모듈을 사용하여 MLflow 모델 레지스트리를 사용하기 시작한다.
  - 모델 버전 관리 및 모델 스테이징을 통해 모델의 라이프사이클을 관리하는 방법도 제공한다.
  - 모뎅 등록

```
# Import MLflow Client from MLflow module
from mlflow import MlflowClient

# Create an instance of MLflow Client Class name client
client = MlflowClient()

# Create new model
client.create_registered_model("Insurance")

# Insurance filter string
insurance_filter_string = "name LIKE 'Insurance%'"

# Search for Insurance models
```

```
print(client.search_registered_models(filter_string=insu
rance_filter_string))

# Not Insurance filter string
not_insurance_filter_string = "name != 'Insurance'"

# Search for models that are not Insurance
print(client.search_registered_models(filter_string=not_
insurance_filter_string))
```

Registering Models

- MLflow 모델 등록
- 모델 수명 주기 관리

```
# Register the first (2022) model
mlflow.register_model("model_2022", "Insurance")

# Register the second (2023) model
mlflow.register_model(f"runs:/{run_id}/model_2023", "Insura
nce")

# Log the model using scikit-learn flavor
mlflow.sklearn.log_model(lr, "model", registered_model_name
="Insurance")
insurance_filter_string = "name = 'Insurance'"

# Search for Insurance models
print(client.search_registered_models(filter_string=insuran
ce_filter_string))
```

Model stages

모델 레지스트리는 모델 버전과 함께 ML 수명 주기를 거치면서 모델의 수명 주기를 관리하는 의미 있는 방법을 제공한다.

- 소프트웨어 환경
  - MLflow 모델 레지스트리는 다양한 소프트웨어 환경에서의 진행과 관련하여 모델 수명 주기를 관리하는 방법을 제공한다.
- MLflow 모델 단계
  - 등록된 모델 버전에 할당할 수 있다.
  - MLflow 는 “없음(모델이 아직 단계를 받지 못함)”, “스테이징(모델이 테스트와 평가를 거칠 때 스테이징이 할당)”, “생산(모델이 모든 테스트를 통과하고 프로덕션에 사용될 준비가 되면 프로덕션이 할당됨)”, “보관됨(모델이 더 이상 사용하지 않아 보관되어야 하는 경우)” 의 미리 정의된 단계를 제공한다.

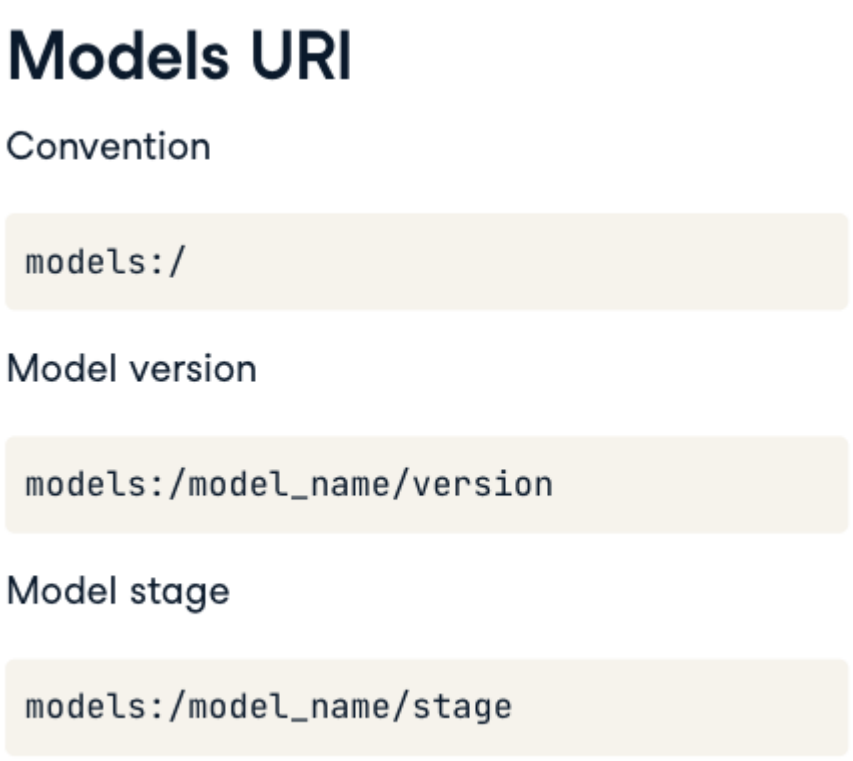
```
# Transition version 2 of Insurance model to stable stage
client.transition_model_version_stage(name="Insurance", ver
sion=2,
        stage="Production"
    )

# Transition version 3 of Insurance model to testing stage
client.transition_model_version_stage(name="Insurance", ver
sion=3,
        stage="Staging"
    )

# Transition version 1 of Insurance model to archive stage
client.transition_model_version_stage(name="Insurance", ver
sion=1,
        stage="Archived"
    )
```

Model deployment

모델 레지스트리를 사용하여 모델을 배포하는 방법

- 모델 배포 방법
  - load_model 함수를 사용하여 모델 레지스트리에서 패치
  - serve 명령을 사용하여 모델 레지스트리에서 제공될 수 있음

- 모델 URI



- 모델 로드
  - 로드 모델 함수를 사용하여 모델 레지스트리에서 모델을 가져오려면 먼저 모델 플레이버를 가져온다.
- 서비스 모델
  - 모델은 MLflow 명령줄 도구의 serve 명령을 사용하여 모델 레지스트리에서 제공할 수 있다.
- 호출 엔드포인트
  - 모델 배포에 serve 명령을 사용할 때 MLflow는 모델을 API 서비스로 제공한다.
  - 기본 포트 5000
- 모델 예측
  - load_model 함수를 사용하거나 제공 명령줄 도구를 사용하여 모델을 배포하는 추론 중에 모델에서 반환 받는다.

```
# Load the Production stage of Insurance model using scikit
-learn flavor
```

```
model = mlflow.sklearn.load_model("models:/Insurance/Production")

# Run prediction on our test data
model.predict(X_test)
```

4. MLflow Projects

Introduction to MLflow Projects

MLflow 프로젝트는 재현 가능한 방식으로 ML 코드를 구성하고 실행하는 방법을 제공하여 ML 라이프사이클을 간소화한다. → 모델 학습, 빌드, 실험 추적, 모델 레지스트리에 등록

- MLproject
  - ml 코드를 포함하는 파일의 디렉토리

  - yaml 파일로 구성

```
"""
# Set name of the Project
name: insurance_model

# Set the environment
python_env: python_env.yaml

entry_points:
```

```
    # Create an entry point
    main:
        # Create a command
        command: 'python3.9 train_model.py'
"""
```

Running MLflow Projects

- Projects API

- MLproject

- 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` or `virtualenv`
- `URI` - URI to MLproject file

```
import mlflow

# Set the run function from the MLflow Projects module
mlflow.projects.run(
    # Set the URI as the current working directory
    uri='./',
    # Set the entry point as main
    entry_point='main',
    # Set the experiment as Insurance
    experiment_name='Insurance',
    env_manager="local",
    synchronous=True,
)
```

Specifying parameters

- MLflow Projects 는 매개변수를 사용하여 유연성과 사용자 정의를 허용한다.
- 매개변수

Parameters block

```
name: project_name
python_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}"
```

- command

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 ./
```

```
"""
name: insurance_model
python_env: python_env.yaml
entry_points:
  main:
    parameters:
      # Create parameter for number of jobs
      n_jobs:
        type: int
        default: 1
      # Create parameter for fit intercept
      fit_intercept:
```

```
        type: bool
        default: True
    # Add parameters to be passed into the command
    command: "python3.9 train_model.py {n_jobs} {fit_interc
ept}"
"""
```

```
import mlflow

# Set the run function from the MLflow Projects module
mlflow.projects.run(
    uri='./',
    entry_point='main',
    experiment_name='Insurance',
    env_manager='local',
    # Set parameters for n_jobs and fit_intercept
    parameters={
        'n_jobs_param': 2,
        'fit_intercept_param': False
    }
)
```

- MLproject for the ML Lifecycle

```
## model engineering

"""
name: insurance_model
python_env: python_env.yaml
entry_points:
  # Set the entry point
  model_engineering:
    parameters:
      # Set n_jobs
      n_jobs:
        type: int
```

```
        default: 1
      # Set fit_intercept
      fit_intercept:
        type: bool
        default: True
    # Pass the parameters to the command
    command: "python3.9 train_model.py {n_jobs} {fit_interc
ept}"
"""

## evaluation
"""
  # Set the model_evaluation entry point
  model_evaluation:
    parameters:
      # Set run_id parameter
      run_id:
        type: str
        default: None
    # Set the parameters in the command
    command: "python3.9 evaluate.py {run_id}"
"""
```

- multi-step workflow

```
## model engineering
# Set run method to model_engineering
model_engineering = mlflow.projects.run(
    uri='./',
    # Set entry point to model_engineering
    entry_point='model_engineering',
    experiment_name='Insurance',
    #  Set the parameters for n_jobs and fit_intercept
    parameters={
        'n_jobs': 2,
        'fit_intercept': False
    },
    env_manager='local'
```

```
)

# Set Run ID of model training to be passed to Model Evalua
tion step
model_engineering_run_id = model_engineering.run_id
print(model_engineering_run_id)


## model evaluation
# Set the MLflow Projects run method
model_evaluation = mlflow.projects.run(
    uri="./",
    # Set the entry point to model_evaluation
    entry_point="model_evaluation",
    # Set the parameter run_id to the run_id output of prev
ious step
    parameters={
        "run_id": model_engineering_run_id,
    },
    env_manager="local"
)

print(model_evaluation.get_status())
```

마무리