Módulo 2: MLFlow Models

Introduction to MLFlow Models

MLFlows Models es un componente utilizado como una manera de estandarizar el empaquetado de los modelos de ML.

Flavors

MLFlow Models usa un concepto llamado **Flavors.** Flavors hace posible escribir herramientas para trabajar con modelos de las librerías más populares usando MLFlow.

Simplifican el proceso de logging, empaquetado y carga de modelos, minimizando la necesidad de escribir código personalizado.

- 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`)
- MXNet Gluon (`gluon`)
- XGBoost (`xgboost`)

```
# Import flavor from mlflow module
import mlflow.FLAVOR
```

Autolog

Algunos de los flavors soportan un método llamado **auto logging,** el cual puede ser usado para de forma automática llevar un registro de métricas, parámetros y modelos en MLFlow Tracking, sin la necesidad de usar declaraciones de logging explíticas.

```
# Automatically log model and metrics
mlflow.FLAVOR.autolog()
```

```
# Scikit-learn built-in flavor
mlflow.sklearn.autolog()
```

Common Metrics

- Regression
  - mean squared error
  - root mean squared error
  - mean absolute error
  - r2 score
- Classification
  - precision score
  - recall score
  - f1 score
  - accuracy score

Common parameters

```
# Train the model
lr = LinearRegression()
lr.fit(X, y)
# Get params
params = lr.get_params(deep=True)
params
```

```
{'copy_X': True, 'fit_intercept': True, 'n_jobs': None,
    'normalize': 'deprecated', 'positive': False}
```

SKlearn Flavor

```python
# Import scikit-learn
import mlflow
from sklearn.linear_model import \
    LinearRegression

# Using auto-logging
mlflow.sklearn.autolog()
```

```python
# Train the model
lr = LinearRegression()
lr.fit(X, y)
```

Model will be logged automatically on `model.fit()`

Storage format

MLFlows empaqueta los modelos usando la siguiente estructura como un formato de almacenamiento estándar.

```
model/
    MLmodel
    conda.yaml
    model.pkl
    python_env.yaml
    requirements.txt
```

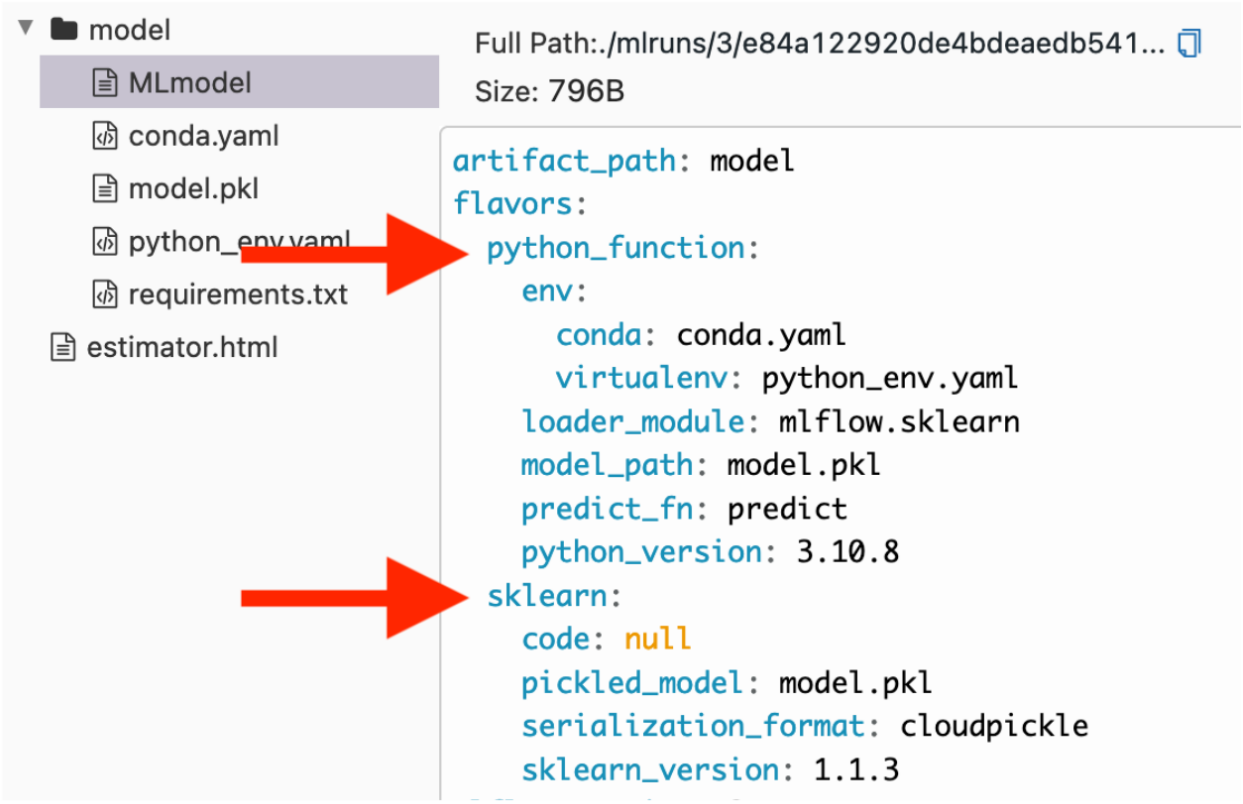
Contents of MLModel

```
artifact_path: model
flavors:
  python_function:
    env:
      conda: conda.yaml
      virtualenv: python_env.yaml
    loader_module: mlflow.sklearn
    model_path: model.pkl
    predict_fn: predict
    python_version: 3.10.8
  sklearn:
    code: null
    pickled_model: model.pkl
    serialization_format: cloudpickle
    sklearn_version: 1.1.3
```

MLModel es un archivo YAML que define la información importante sobre el modelo.

YAML es un formato de serialización leíble para humanos y usado para archivos de configuración.

Podemos ver que el archivo MLModel contiene dos Flavors: sklearn y python_function; eso significa que podemos cargar el modelo usando cualquiera de los dos.



Model API

MLFlow REST API

MLFlow usa una API REST que permite a los usuarios crear, listar y recuperar información de cualquier componente de MLFlow.

Model API

Con ella, los usuarios pueden guardar, registrar y cargar un modelo de MLFlow usando un Flavor particular.

Model API functions

```
# Save a model to the local filesystem
mlflow.sklearn.save_model(model, path)
```

```
# Log a model as an artifact to MLflow Tracking.
mlflow.sklearn.log_model(model, artifact_path)
```

```
# Load a model from local filesystem or from MLflow Tracking.
mlflow.sklearn.load_model(model_uri)
```

Load model

- Local Filesystem - `relative/path/to/local/model` or `/Users/me/path/to/local/model`
- MLflow Tracking - `runs:/<mlflow_run_id>/run-relative/path/to/model`
- S3 Support - `s3://my_bucket/path/to/model`

Save model

```
# Model
lr = LogisticRegression()
lr.fit(X, y)

# Save model locall
mlflow.sklearn.save_model(lr, "local_path")
```

```
ls local_path/
```

```
MLmodel             model.pkl        requirements.txt
conda.yaml          python_env.yaml
```

Load local model

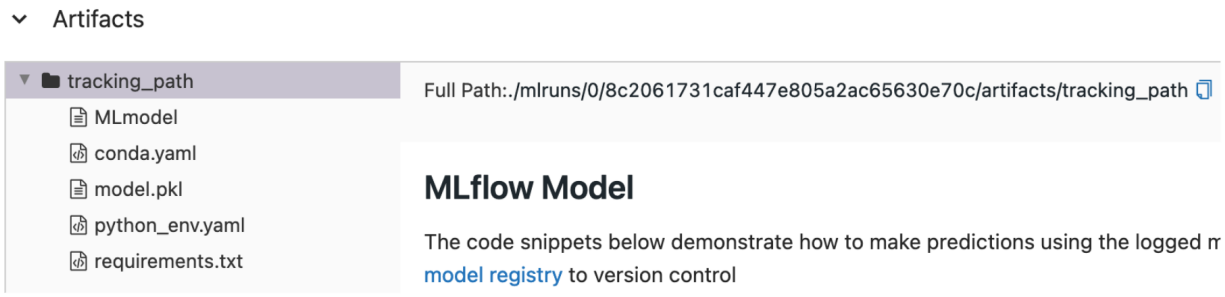
```
# Load model from local path
model = mlflow.sklearn.load_model("local_path")

# Show model
model
```

```
LogisticRegression()
```

Log model

Usar log_model en lugar el del autolog permite especificar opciones adicionales como un nombre o una ruta de artefacto.

```
# Model
lr = LogisticRegression(n_jobs=n_jobs)
lr.fit(X, y)

# Log model
mlflow.sklearn.log_model(lr, "tracking_path")
```



Last run id

```
# Get last active run
run = mlflow.last_active_run()
# Set run_id variable
run_id = run.info.run_id
run_id
```

Load model from MLFlow Tracking

```
# Pass run_id as f-string literal
model = mlflow.sklearn.load_model(f"runs:/{run_id}/tracking_path")
# Show model
model
```

```
LogisticRegression()
```

Custom models

Es imposible cubrir todos los casos de uso. MLFlow tiene una solución para esto.

Custom Python Models

MLFlow proporciona algo llamado **Model Customization** usando modelos de Python para permitir a los usuarios personalizar modelos que se ajusten a sus casos de uso. Es un Built in Flavor llamado **python_function. mlflow.pyfunc** proporciona muchas de las mismas funciones: load_model(), save_model(), log_model()

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

Example custom Class

```python
import mlflow.pyfunc

# Define the model class
class CustomPredict(mlflow.pyfunc.PythonModel):
    # Load artifacts
    def load_context(self, context):
        self.model = mlflow.sklearn.load_model(context.artifacts["custom_model"])
    # Evaluate input using custom_function()
    def predict(self, context, model_input):
        prediction = self.model.predict(model_input)
        return custom_function(prediction)
```

Saving and logging a custom model

```python
# Save model to local filesystem
mlflow.pyfunc.save_model(path="custom_model", python_model=CustomPredict())
```

```python
# Log model to MLflow Tracking
mlflow.pyfunc.log_model(artifact_path="custom_model", python_model=CustomPredict())
```

Loading custom models

```
# Load model from local filesystem
mlflow.pyfunc.load_model("local")
```

```
# Load model from MLflow Tracking
mlflow.pyfunc.load_model("runs:/run_id/tracking_path")
```

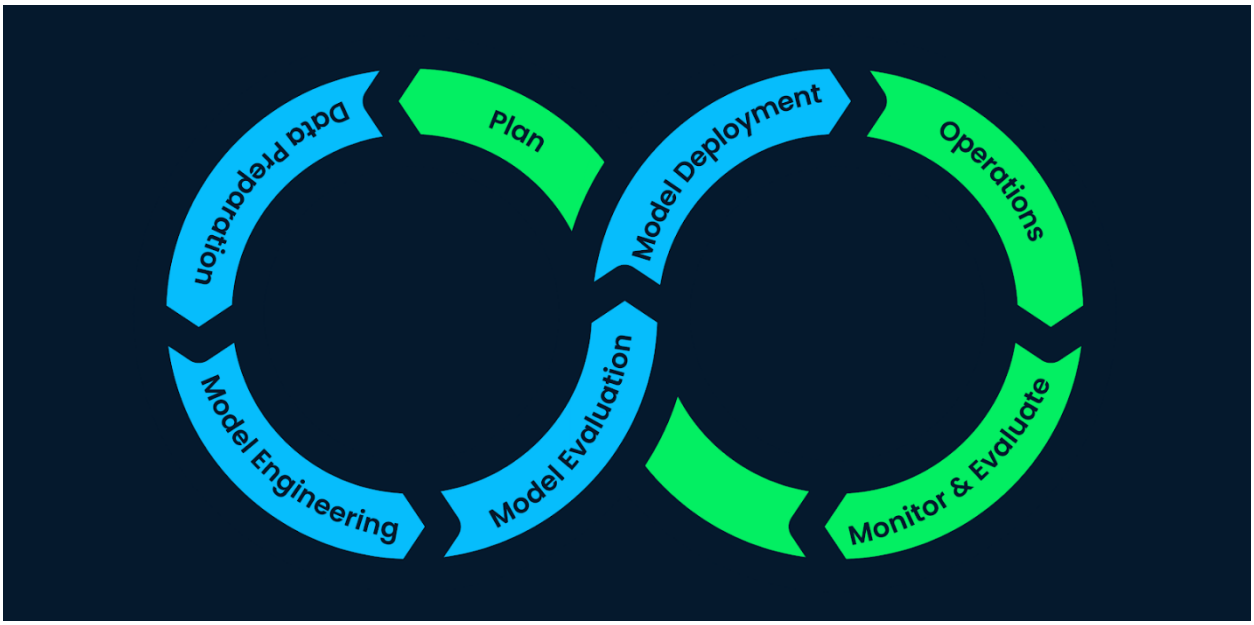
Model Evalation

```
# Training Data
X_train, X_test, y_train, y_test = \
    train_test_split(X, y,
    train_size=0.7,random_state=0)

# Linear Regression model
lr = LinearRegression()
lr.fit(X_train, y_train)
```

```
# Dataset
eval_data = X_test
eval_data["test_label"] = y_test

# Evaluate model with Dataset
mlflow.evaluate(
    "runs:/run_id/model",
    eval_data,
    targets="test_label",
    model_type="regressor"
)
```

Model serving

Model Deployment



Como el empaquetado del modelo es estandarizado, MLFlow Models también permite simplificar el despliegue del modelo.

REST API

MLFlow sirve los modelos como una API REST. Se tienen 4 endpoints:

- `/ping` - for health checks
- `/health` - for health checks
- `/version` - for getting the version of MLflow
- `/invocations` - for model scoring
- Port 5000

Invocations endpoint

Acepta CSV o JSON. Necesita un **content-type header** para especificar el formato.

`/invocations`

```
No,Name,Subject
1,Bill Johnson,English
2,Gary Valentine,Mathematics
```

`Content-Type` : `application/json` or `application/csv`

```
{
    "1": {
        "No": "1",
        "Name": "Bill Johnson",
        "Subject": "English"
    },
    "2": {
        "No": "2",
        "Name": "Gary Valentine",
        "Subject": "Mathematics"
    }
}
```

CSV format

- Pandas Dataframe
- `pandas_df.to_csv()`

JSON format

- `dataframe_split` - pandas DataFrame in split orientation
- `dataframe_records` - pandas DataFrame in records orientation

DataFrame split

```
# Dataframe split orientation
{
  "dataframe_split": {
      "columns": ["sex", "age", "weight"],
      "data": [["male", 23, 160], ["female", 33, 120]]
  }
}
```

Serving Models

Para servir un modelo, MLFlow incluye un comando llamado **serve.** Se usa para lanzar un sitio web local que ejecuta la API REST usada para servir el modelo.

```
# Local Filesystem
mlflow models serve -m relative/path/to/local/model
```

```
# Run ID
mlflow models serve -m runs:/<mlflow_run_id>/artifacts/model
```

```
# AWS S3
mlflow models serve -m s3://my_bucket/path/to/model
```

Example

```
# Serve model from run
mlflow models serve -m runs:/e84a122920de4bdeaedb54146deeb429/artifacts/model
```

```
2023/03/12 16:28:28 INFO mlflow.models.flavor_backend_registry:
Selected backend for flavor 'python_function'
2023/03/12 16:28:28 INFO mlflow.pyfunc.backend: === Running command
'exec gunicorn --timeout=60 -b 127.0.0.1:5000 -w 1 ${GUNICORN_CMD_ARGS} --
mlflow.pyfunc.scoring_server.wsgi:app'
[2023-03-12 16:28:29 -0400] [48431] [INFO] Starting gunicorn 20.1.0
[2023-03-12 16:28:29 -0400] [48431] [INFO] Listening at: http://127.0.0.1:5000
(48431)
[2023-03-12 16:28:29 -0400] [48431] [INFO] Using worker: sync
[2023-03-12 16:28:29 -0400] [48432] [INFO] Booting worker with pid: 48432
```

```
# Send dataframe_split orientation payload to MLflow
curl http://127.0.0.1:5000/invocations -H 'Content-Type: application/json' -d '{
  "dataframe_split": {
      "columns": ["sex", "age", "weight"],
      "data": [["male", 23, 160], ["female", 33, 120]]
  }
}'
```

```
{"predictions": [1, 0]}
```