Log models with MLflow

You can use MLflow in Azure Machine Learning to log models. When you log a model as a model instead of an artifact, a MLmodel is created in the output directory. The MLmodel file contains all the model's metadata. You can customize the model's signature when logging the model.

Before you start

You'll need the latest version of the **azureml-ai-ml** package to run the code in this notebook. Run the cell below to verify that it is installed.

> **Note:** If the **azure-ai-ml** package is not installed, run `pip install azure-ai-ml` to install it.

In [ ]:

```
pip show azure-ai-ml
```

Connect to your workspace

With the required SDK packages installed, now you're ready to connect to your workspace.

To connect to a workspace, we need identifier parameters - a subscription ID, resource group name, and workspace name. Since you're working with a compute instance, managed by Azure Machine Learning, you can use the default values to connect to the workspace.

In [ ]:

```
from azure.identity import DefaultAzureCredential, InteractiveBrowserCredential
from azure.ai.ml import MLClient

try:
    credential = DefaultAzureCredential()
    # Check if given credential can get token successfully.
    credential.get_token("https://management.azure.com/.default")
except Exception as ex:
    # Fall back to InteractiveBrowserCredential in case DefaultAzureCredential not work
    credential = InteractiveBrowserCredential()
```

In [ ]:

```
# Get a handle to workspace
ml_client = MLClient.from_config(credential=credential)
```

Autologging with MLflow

When you use autologging, your model is automatically logged. The model flavor and schema is inferred.

Run the following cell to create the **train-model-autolog.py** script in the **src** folder. The script trains a classification model by using the **diabetes.csv** file in the same folder, which is passed as an argument.

In [ ]:

```
import os

# create a folder for the script files
script_folder = 'src'
os.makedirs(script_folder, exist_ok=True)
print(script_folder, 'folder created')
```

In [ ]:

```
%%writefile $script_folder/train-model-autolog.py
# import libraries
import mlflow
import argparse
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import roc_auc_score
from sklearn.metrics import roc_curve
import matplotlib.pyplot as plt

def main(args):
    # enable autologging
    mlflow.autolog()

    # read data
    df = get_data(args.training_data)

    # split data
    X_train, X_test, y_train, y_test = split_data(df)

    # train model
    model = train_model(args.reg_rate, X_train, X_test, y_train, y_test)

    eval_model(model, X_test, y_test)

# function that reads the data
def get_data(path):
    print("Reading data...")
    df = pd.read_csv(path)

    return df

# function that splits the data
def split_data(df):
    print("Splitting data...")
    X, y = df[['Pregnancies','PlasmaGlucose','DiastolicBloodPressure','TricepsThickness'
,
    'SerumInsulin','BMI','DiabetesPedigree','Age']].values, df['Diabetic'].values

    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.30, random_sta
te=0)

    return X_train, X_test, y_train, y_test

# function that trains the model
def train_model(reg_rate, X_train, X_test, y_train, y_test):
    mlflow.log_param("Regularization rate", reg_rate)
    print("Training model...")
    model = LogisticRegression(C=1/reg_rate, solver="liblinear").fit(X_train, y_train)

    return model

# function that evaluates the model
def eval_model(model, X_test, y_test):
    # calculate accuracy
    y_hat = model.predict(X_test)
    acc = np.average(y_hat == y_test)
    print('Accuracy:', acc)

def parse_args():
    # setup arg parser
    parser = argparse.ArgumentParser()

    # add arguments
    parser.add_argument("--training_data", dest='training_data',
                        type=str)
    parser.add_argument("--reg_rate", dest='reg_rate',
                        type=float, default=0.01)
    # parse args
    args = parser.parse_args()

    # return args
    return args

# run script
if __name__ == "__main__":
    # add space in logs
    print("\n\n")
    print("*" * 60)

    # parse args
    args = parse_args()

    # run main function
    main(args)

    # add space in logs
    print("*" * 60)
    print("\n\n")
```

Now, you can submit the script as a command job.

Run the cell below to train the model.

In [ ]:

```
from azure.ai.ml import command

# configure job

job = command(
    code="./src",
    command="python train-model-autolog.py --training_data diabetes.csv",
    environment="AzureML-sklearn-0.24-ubuntu18.04-py37-cpu@latest",
    compute="aml-cluster",
    display_name="diabetes-train-autolog",
    experiment_name="diabetes-training"
    )

# submit job
returned_job = ml_client.create_or_update(job)
aml_url = returned_job.studio_url
print("Monitor your job at", aml_url)
```

In the Studio, navigate to the diabetes-train-autolog job to explore the overview of the command job you ran. Find the logged artifacts in the Outputs + logs tab. Select the `model` folder to find the `MLmodel` file and explore its contents.

Specify the flavor with autologging

Ex., scikit-learn is the flavor of the model in this example.

You can use autologging, but still specify the flavor of the model. In the example, the model's flavor is scikit-learn.

Run the following cell to create the train-model-sklearn.py script in the src folder. The script trains a classification model by using the diabetes.csv file in the same folder, which is passed as an argument.

In [ ]:

```
%%writefile $script_folder/train-model-sklearn.py
# import libraries
import mlflow
import argparse
import pandas as pd
import numpy as np
```

```python
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import roc_auc_score
from sklearn.metrics import roc_curve
import matplotlib.pyplot as plt

def main(args):
    # enable autologging
    mlflow.sklearn.autolog()

    # read data
    df = get_data(args.training_data)

    # split data
    X_train, X_test, y_train, y_test = split_data(df)

    # train model
    model = train_model(args.reg_rate, X_train, X_test, y_train, y_test)

    eval_model(model, X_test, y_test)

# function that reads the data
def get_data(path):
    print("Reading data...")
    df = pd.read_csv(path)

    return df

# function that splits the data
def split_data(df):
    print("Splitting data...")
    X, y = df[['Pregnancies','PlasmaGlucose','DiastolicBloodPressure','TricepsThickness'
,
    'SerumInsulin','BMI','DiabetesPedigree','Age']].values, df['Diabetic'].values

    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.30, random_sta
te=0)

    return X_train, X_test, y_train, y_test

# function that trains the model
def train_model(reg_rate, X_train, X_test, y_train, y_test):
    mlflow.log_param("Regularization rate", reg_rate)
    print("Training model...")
    model = LogisticRegression(C=1/reg_rate, solver="liblinear").fit(X_train, y_train)

    return model

# function that evaluates the model
def eval_model(model, X_test, y_test):
    # calculate accuracy
    y_hat = model.predict(X_test)
    acc = np.average(y_hat == y_test)
    print('Accuracy:', acc)

    # calculate AUC
    y_scores = model.predict_proba(X_test)
    auc = roc_auc_score(y_test,y_scores[:,1])
    print('AUC: ' + str(auc))

    # plot ROC curve
    fpr, tpr, thresholds = roc_curve(y_test, y_scores[:,1])
    fig = plt.figure(figsize=(6, 4))
    # Plot the diagonal 50% line
    plt.plot([0, 1], [0, 1], 'k--')
    # Plot the FPR and TPR achieved by our model
    plt.plot(fpr, tpr)
    plt.xlabel('False Positive Rate')
    plt.ylabel('True Positive Rate')
    plt.title('ROC Curve')
    plt.savefig("ROC-Curve.png")
```

```
def parse_args():
    # setup arg parser
    parser = argparse.ArgumentParser()

    # add arguments
    parser.add_argument("--training_data", dest='training_data',
                        type=str)
    parser.add_argument("--reg_rate", dest='reg_rate',
                        type=float, default=0.01)

    # parse args
    args = parser.parse_args()

    # return args
    return args

# run script
if __name__ == "__main__":
    # add space in logs
    print("\n\n")
    print("*" * 60)

    # parse args
    args = parse_args()

    # run main function
    main(args)

    # add space in logs
    print("*" * 60)
    print("\n\n")
```

Now, you can submit the script as a command job.

Run the cell below to train the model.

In [ ]:

```
from azure.ai.ml import command

# configure job

job = command(
    code="./src",
    command="python train-model-sklearn.py --training_data diabetes.csv",
    environment="AzureML-sklearn-0.24-ubuntu18.04-py37-cpu@latest",
    compute="aml-cluster",
    display_name="diabetes-train-sklearn",
    experiment_name="diabetes-training"
    )

# submit job
returned_job = ml_client.create_or_update(job)
aml_url = returned_job.studio_url
print("Monitor your job at", aml_url)
```

In the Studio, navigate to the diabetes-train-sklearn job to explore the overview of the command job you ran. Find the logged artifacts in the Outputs + logs tab. Select the `model` folder to find the `MLmodel` file and explore its contents.

Compare the `MLmodel` files of the previous two runs. You'll notice that they're the same, indicating that MLflow's autolog feature correctly inferred the model's flavor.

Customize the model with an inferred signature

You can manually log the model when using autologging. By using 'log_models=False', autologging will not log the model. You'll create a signature by inferring it from the training dataset and predicted results. And finally, you'll log the scikit-learn model.

Run the following cell to create the train-model-infer.py script in the src folder. The script trains a classification model by using the diabetes.csv file in the same folder, which is passed as an argument.

In [ ]:

```
%%writefile $script_folder/train-model-infer.py
# import libraries
import mlflow
import argparse
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import roc_auc_score
from sklearn.metrics import roc_curve
import matplotlib.pyplot as plt
import mlflow.sklearn
from mlflow.models.signature import infer_signature

def main(args):
    # enable autologging
    mlflow.autolog(log_models=False)

    # read data
    df = get_data(args.training_data)

    # split data
    X_train, X_test, y_train, y_test = split_data(df)

    # train model
    model = train_model(args.reg_rate, X_train, X_test, y_train, y_test)

    # evaluate model
    y_hat = eval_model(model, X_test, y_test)

    # create the signature by inferring it from the datasets
    signature = infer_signature(X_train, y_hat)

    # manually log the model
    mlflow.sklearn.log_model(model, "model", signature=signature)

# function that reads the data
def get_data(path):
    print("Reading data...")
    df = pd.read_csv(path)

    return df

# function that splits the data
def split_data(df):
    print("Splitting data...")
    X, y = df[['Pregnancies','PlasmaGlucose','DiastolicBloodPressure','TricepsThickness'
,
    'SerumInsulin','BMI','DiabetesPedigree','Age']].values, df['Diabetic'].values

    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.30, random_sta
te=0)

    return X_train, X_test, y_train, y_test

# function that trains the model
def train_model(reg_rate, X_train, X_test, y_train, y_test):
    mlflow.log_param("Regularization rate", reg_rate)
    print("Training model...")
    model = LogisticRegression(C=1/reg_rate, solver="liblinear").fit(X_train, y_train)

    return model

# function that evaluates the model
def eval_model(model, X_test, y_test):
    # calculate accuracy
    y_hat = model.predict(X_test)
    acc = np.average(y_hat == y_test)
    print('Accuracy:', acc)

    return y_hat

def parse_args():
    # setup arg parser
    parser = argparse.ArgumentParser()

    # add arguments
    parser.add_argument("--training_data", dest='training_data',
                        type=str)
    parser.add_argument("--reg_rate", dest='reg_rate',
                        type=float, default=0.01)

    # parse args
    args = parser.parse_args()

    # return args
    return args

# run script
if __name__ == "__main__":
    # add space in logs
    print("\n\n")
    print("*" * 60)

    # parse args
    args = parse_args()

    # run main function
    main(args)

    # add space in logs
    print("*" * 60)
    print("\n\n")
```

Now, you can submit the script as a command job.

Run the cell below to train the model.

In [ ]:

```
from azure.ai.ml import command

# configure job

job = command(
    code="./src",
    command="python train-model-infer.py --training_data diabetes.csv",
    environment="AzureML-sklearn-0.24-ubuntu18.04-py37-cpu@latest",
    compute="aml-cluster",
    display_name="diabetes-train-infer",
    experiment_name="diabetes-training"
    )

# submit job
returned_job = ml_client.create_or_update(job)
aml_url = returned_job.studio_url
print("Monitor your job at", aml_url)
```

In the Studio, navigate to the diabetes-train-infer job to explore the overview of the command job you ran. Find the logged artifacts in the Outputs + logs tab. Select the `model` folder to find the `MLmodel` file and explore its contents.

Compare the `MLmodel` files with the previous two runs. You'll notice that they're all the same, indicating that MLflow's autolog feature correctly inferred the model's signature too.

Customize the model with a defined signature

You can manually log the model when using autologging. By using 'log_models=False', autologging will not log the model. You'll create a signature by inferring it from the training dataset and predicted results. And finally, you'll log the scikit-learn model.

Run the following cell to create the train-model-infer.py script in the src folder. The script trains a classification model by using the diabetes.csv file in the same folder, which is passed as an argument.

In [ ]:

Signatures are indicated when the model gets logged and persisted in the MLmodel file, in the signature section. Autolog's feature in MLflow automatically infers signatures in a best effort way.

```
%%writefile $script_folder/train-model-signature.py
# import libraries
import mlflow
import argparse
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import roc_auc_score
from sklearn.metrics import roc_curve
import matplotlib.pyplot as plt
import mlflow.sklearn
from mlflow.models.signature import ModelSignature
from mlflow.types.schema import Schema, ColSpec

def main(args):
    # enable autologging
    mlflow.autolog(log_models=False)

    # read data
    df = get_data(args.training_data)

    # split data
    X_train, X_test, y_train, y_test = split_data(df)

    # train model
    model = train_model(args.reg_rate, X_train, X_test, y_train, y_test)

    # evaluate model
    y_hat = eval_model(model, X_test, y_test)

    # create the signature manually
    input_schema = Schema([
    ColSpec("integer", "Pregnancies"),
    ColSpec("integer", "PlasmaGlucose"),
    ColSpec("integer", "DiastolicBloodPressure"),
    ColSpec("integer", "TricepsThickness"),
    ColSpec("integer", "DiastolicBloodPressure"),
    ColSpec("integer", "SerumInsulin"),
    ColSpec("double", "BMI"),
    ColSpec("double", "DiabetesPedigree"),
    ColSpec("integer", "Age"),
    ])

    output_schema = Schema([ColSpec("boolean")])

    # Create the signature object
    signature = ModelSignature(inputs=input_schema, outputs=output_schema)

    # manually log the model
    mlflow.sklearn.log_model(model, "model", signature=signature)

# function that reads the data
def get_data(path):
    print("Reading data...")
    df = pd.read_csv(path)

    return df
```

```
# function that splits the data
def split_data(df):
    print("Splitting data...")
    X, y = df[['Pregnancies','PlasmaGlucose','DiastolicBloodPressure','TricepsThickness'
,
    'SerumInsulin','BMI','DiabetesPedigree','Age']].values, df['Diabetic'].values

    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.30, random_sta
te=0)

    return X_train, X_test, y_train, y_test

# function that trains the model
def train_model(reg_rate, X_train, X_test, y_train, y_test):
    mlflow.log_param("Regularization rate", reg_rate)
    print("Training model...")
    model = LogisticRegression(C=1/reg_rate, solver="liblinear").fit(X_train, y_train)

    return model

# function that evaluates the model
def eval_model(model, X_test, y_test):
    # calculate accuracy
    y_hat = model.predict(X_test)
    acc = np.average(y_hat == y_test)
    print('Accuracy:', acc)

    return y_hat

def parse_args():
    # setup arg parser
    parser = argparse.ArgumentParser()

    # add arguments
    parser.add_argument("--training_data", dest='training_data',
                        type=str)
    parser.add_argument("--reg_rate", dest='reg_rate',
                        type=float, default=0.01)

    # parse args
    args = parser.parse_args()

    # return args
    return args

# run script
if __name__ == "__main__":
    # add space in logs
    print("\n\n")
    print("*" * 60)

    # parse args
    args = parse_args()

    # run main function
    main(args)

    # add space in logs
    print("*" * 60)
    print("\n\n")
```

Now, you can submit the script as a command job.

Run the cell below to train the model.

In [ ]:

```
from azure.ai.ml import command

# configure job
```

```
job = command(
    code="./src",
    command="python train-model-signature.py --training_data diabetes.csv",
    environment="AzureML-sklearn-0.24-ubuntu18.04-py37-cpu@latest",
    compute="aml-cluster",
    display_name="diabetes-train-signature",
    experiment_name="diabetes-training"
    )

# submit job
returned_job = ml_client.create_or_update(job)
aml_url = returned_job.studio_url
print("Monitor your job at", aml_url)
```

In the Studio, navigate to the diabetes-train-signature job to explore the overview of the command job you ran. Find the logged artifacts in the Outputs + logs tab. Select the `model` folder to find the `MLmodel` file and explore its contents.

Compare the `MLmodel` files with the previous runs. You'll notice that the signature is different from the previous runs. Previous runs used tensor-based signatures, whereas the latest run used a column-based signature.

Register the model

When you choose a model you want to deploy, you can first register the model.

To register the latest model, you'll refer to the name of the job run. By registering the model as an MLflow model, you can easily deploy it later.

In [ ]:

```
from azure.ai.ml.entities import Model
from azure.ai.ml.constants import AssetTypes

job_name = returned_job.name

run_model = Model(
    path=f"azureml://jobs/{job_name}/outputs/artifacts/paths/model/",
    name="mlflow-diabetes",
    description="Model created from run.",
    type=AssetTypes.MLFLOW_MODEL,
)
# Uncomment after adding required details above
ml_client.models.create_or_update(run_model)
```

In the Studio, navigate to the Models page. In the model list, find the `mlflow-diabetes` model and select it to explore it further.

- **In the Details tab of the `mlflow-diabetes` model, you can review that it's a `MLFLOW` type model and the job that trained the model.**
- **In the Artifacts tab you can find the directory with the `MLmodel` file.**

If you want to explore the model's behavior further, you can optionally choose to deploy the model to a real-time endpoint.