AzureML in a Day

Learn how a data scientist uses Azure Machine Learning (Azure ML) to train a model, then use the model for prediction. This tutorial will help you become familiar with the core concepts of Azure ML and their most common usage.

You'll learn how to submit a *command job* to run your *training script*, configured with the *job environment* necessary to run the script.

The training script handles the data preparation, then trains and registers a model. Once you have the model, you'll *deploy* it as an *endpoint*, then call the endpoint for *inferencing*.

The steps you'll take are:

- Connect to your Azure ML workspace
- Create your job environment
- Create your training script
- Create and run your command job to run the training script, configured with the appropriate job environment
- View the output of your training script
- Deploy the newly-trained model as an endpoint
- Call the Azure ML endpoint for inferencing

Prerequisites

- An Azure subscription. If you don't have an Azure subscription, create a free account before you begin.
- A working Azure ML workspace. A workspace can be created via Azure Portal, Azure CLI, or Python SDK. Read more.
- An Azure Machine Learning workspace
- A workspace and compute instance which you can create by completing the Quickstart: Get started with Azure Machine Learning

Connect to the workspace

Before you dive in the code, you'll need to connect to your Azure ML workspace. The workspace is the top-level resource for Azure Machine Learning, providing a centralized place to work with all the artifacts you create when you use Azure Machine Learning.

We're using `DefaultAzureCredential` to get access to workspace. `DefaultAzureCredential` is used to handle most Azure SDK authentication scenarios.

Reference for more available credentials if it doesn't work for you: configure credential example, azure-identity reference doc.

```
# Handle to the workspace
from azure.ai.ml import MLClient
```

Handles most of the azure sdk authentication scenarios.

```
# Authentication package
from azure.identity import DefaultAzureCredential

credential = DefaultAzureCredential()
```

If you want to use a browser to login and authenticate, you can use the following code instead. In this example, you'll use the `DefaultAzureCredential`.

```
# Handle to the workspace
# from azure.ai.ml import MLClient

# Authentication package
# from azure.identity import InteractiveBrowserCredential
# credential = InteractiveBrowserCredential()
```

In the next cell, enter your Subscription ID, Resource Group name and Workspace name. To find these values:

1. In the upper right Azure Machine Learning studio toolbar, select your workspace name.
2. Copy the value for workspace, resource group and subscription ID into the code.

3. You'll need to copy one value, close the area and paste, then come back for the next one.

image of workspace credentials

```
# Get a handle to the workspace
ml_client = MLClient(
    credential=credential,
    subscription_id="<SUBSCRIPTION_ID>",
    resource_group_name="<RESOURCE_GROUP>",
    workspace_name="<AML_WORKSPACE_NAME>",
)
```

The result is a handler to the workspace that you'll use to manage other resources and jobs.

> [!IMPORTANT] Creating MLClient will not connect to the workspace. The client initialization is lazy, it will wait for the first time it needs to make a call (in the notebook below, that will happen during job environment creation).

Create a job environment

To run your AzureML job, you'll need an environment. An environment lists the software runtime and libraries that you want installed on the compute where you'll be training. It's similar to your Python environment on your local machine.

AzureML provides many curated or ready-made environments, which are useful for common training and inference scenarios. You can also create your own custom environments using a docker image, or a conda configuration. Conda yaml file.

In this example, you'll create a custom conda environment for your jobs, using a conda yaml file.

First, create a directory to store the file in.

```
import os

dependencies_dir = "./dependencies"
os.makedirs(dependencies_dir, exist_ok=True)
```

Now, create the file in the dependencies directory. The cell below uses IPython magic to write the file into the directory you just created.

```
%%writefile {dependencies_dir}/conda.yaml
name: model-env
channels:
  - conda-forge
dependencies:
  - python=3.8
  - numpy=1.21.2
  - pip=21.2.4
  - scikit-learn=0.24.2
  - scipy=1.7.1
  - pandas>=1.1,<1.2
  - pip:
    - inference-schema[numpy-support]==1.3.0
    - xlrd==2.0.1
    - mlflow== 1.26.1
    - azureml-mlflow==1.42.0
    - psutil>=5.8,<5.9
    - tqdm>=4.59,<4.60
    - ipykernel~=6.0
    - matplotlib
```

The specification contains some usual packages, that you'll use in your job (numpy, pip).

Reference this *yaml* file to create and register this custom environment in your workspace:

```
from azure.ai.ml.entities import Environment

custom_env_name = "aml-scikit-learn"

pipeline_job_env = Environment(
    name=custom_env_name,
    description="Custom environment for Credit Card Defaults
pipeline",
    tags={"scikit-learn": "0.24.2"},
```

```
    conda_file=os.path.join(dependencies_dir, "conda.yaml"),
    image="mcr.microsoft.com/azureml/openmpi3.1.2-ubuntu18.04:latest",
)
pipeline_job_env =
ml_client.environments.create_or_update(pipeline_job_env)

print(
    f"Environment with name {pipeline_job_env.name} is registered to
workspace, the environment version is {pipeline_job_env.version}"
)
```

What is a command job?

You'll create an Azure ML *command job* to train a model for credit default prediction. The command job is used to run a *training script* in a specified environment on serverless compute. You've already created the environment. Next you'll create the training script.

The *training script* handles the data preparation, training and registering of the trained model. In this tutorial, you'll create a Python training script.

Command jobs can be run from CLI, Python SDK, or studio interface. In this tutorial, you'll use the Azure ML Python SDK v2 to create and run the command job.

After running the training job, you'll deploy the model, then use it to produce a prediction.

Create training script

Let's start by creating the training script - the *main.py* Python file.

First create a source folder for the script:

```
import os

train_src_dir = "./src"
os.makedirs(train_src_dir, exist_ok=True)
```

This script handles the preprocessing of the data, splitting it into test and train data. It then consumes this data to train a tree based model and return the output model.

MLFlow will be used to log the parameters and metrics during our pipeline run.

The cell below uses IPython magic to write the training script into the directory you just created.

```
%%writefile {train_src_dir}/main.py
import os
import argparse
import pandas as pd
import mlflow
import mlflow.sklearn
from sklearn.ensemble import GradientBoostingClassifier
```

```
from sklearn.metrics import classification_report
from sklearn.model_selection import train_test_split

def main():
    """Main function of the script."""

    # input and output arguments
    parser = argparse.ArgumentParser()
    parser.add_argument("--data", type=str, help="path to input data")
    parser.add_argument("--test_train_ratio", type=float,
required=False, default=0.25)
    parser.add_argument("--n_estimators", required=False, default=100,
type=int)
    parser.add_argument("--learning_rate", required=False,
default=0.1, type=float)
    parser.add_argument("--registered_model_name", type=str,
help="model name")
    args = parser.parse_args()

    # Start Logging
    mlflow.start_run()

    # enable autologging
    mlflow.sklearn.autolog()

    ####################
    #<prepare the data>
    ####################
    print(" ".join(f"{k}={v}" for k, v in vars(args).items()))

    print("input data:", args.data)

    credit_df = pd.read_excel(args.data, header=1, index_col=0)

    mlflow.log_metric("num_samples", credit_df.shape[0])
    mlflow.log_metric("num_features", credit_df.shape[1] - 1)

    train_df, test_df = train_test_split(
        credit_df,
        test_size=args.test_train_ratio,
    )
    ####################
    #</prepare the data>
    ####################

    ##################
    #<train the model>
    ##################
    # Extracting the label column
```

```
    y_train = train_df.pop("default payment next month")

    # convert the dataframe values to array
    X_train = train_df.values

    # Extracting the label column
    y_test = test_df.pop("default payment next month")

    # convert the dataframe values to array
    X_test = test_df.values

    print(f"Training with data of shape {X_train.shape}")

    clf = GradientBoostingClassifier(
        n_estimators=args.n_estimators,
learning_rate=args.learning_rate
    )
    clf.fit(X_train, y_train)

    y_pred = clf.predict(X_test)

    print(classification_report(y_test, y_pred))
    ###################
    #</train the model>
    ###################

    ##########################
    #<save and register model>
    ##########################
    # Registering the model to the workspace
    print("Registering the model via MLFlow")
    mlflow.sklearn.log_model(
        sk_model=clf,
        registered_model_name=args.registered_model_name,
        artifact_path=args.registered_model_name,
    )

    # Saving the model to a file
    mlflow.sklearn.save_model(
        sk_model=clf,
        path=os.path.join(args.registered_model_name,
"trained_model"),
    )
    ##########################
    #</save and register model>
    ##########################

    # Stop Logging
    mlflow.end_run()
```

```
if __name__ == "__main__":
    main()
```

As you can see in this script, once the model is trained, the model file is saved and registered to the workspace. Now you can use the registered model in inferencing endpoints.

Configure the command

Now that you have a script that can perform the desired tasks, you'll use the general purpose **command** that can run command line actions. This command line action can be directly calling system commands or by running a script.

Here, you'll create input variables to specify the input data, split ratio, learning rate and registered model name. The command script will:

- Use the environment created earlier - you can use the `@latest` notation to indicate the latest version of the environment when the command is run.
- Configure some metadata like display name, experiment name etc. An *experiment* is a container for all the iterations you do on a certain project. All the jobs submitted under the same experiment name would be listed next to each other in Azure ML studio.
- Configure the command line action itself - `python main.py` in this case. The inputs/outputs are accessible in the command via the `${{ ... }}` notation.
- In this sample, we access the data from a file on the internet.

```
from azure.ai.ml import command
from azure.ai.ml import Input

registered_model_name = "credit_defaults_model"

job = command(
    inputs=dict(
        data=Input(
            type="uri_file",
            path="https://archive.ics.uci.edu/ml/machine-learning-
databases/00350/default%20of%20credit%20card%20clients.xls",
        ),
        test_train_ratio=0.2,
        learning_rate=0.25,
        registered_model_name=registered_model_name,
    ),
    code="./src/",  # location of source code
    command="python main.py --data ${{inputs.data}} --test_train_ratio
${{inputs.test_train_ratio}} --learning_rate ${{inputs.learning_rate}}
--registered_model_name ${{inputs.registered_model_name}}",
    environment="aml-scikit-learn@latest",
    experiment_name="train_model_credit_default_prediction",
```

```
    display_name="credit_default_prediction",
)
```

Submit the job

It's now time to submit the job to run in AzureML. This time you'll use `create_or_update` on `ml_client.jobs`.

```
ml_client.create_or_update(job)
```

View job output and wait for job completion

View the job in Azure ML studio by selecting the link in the output of the previous cell.

The output of this job will look like this in Azure ML studio. Explore the tabs for various details like metrics, outputs etc. Once completed, the job will register a model in your workspace as a result of training.

Screenshot that shows the job overview

> [!IMPORTANT] Wait until the status of the job is complete before returning to this notebook to continue. The job will take 2 to 3 minutes to run. It could take longer (up to 10 minutes) if the compute has been scaled down to zero nodes and custom environment is still building.

Deploy the model as an online endpoint

Now deploy your machine learning model as a web service in the Azure cloud, an `online endpoint`.

To deploy a machine learning service, you usually need:

- The model assets (file, metadata) that you want to deploy. You've already registered these assets in your training job.
- Some code to run as a service. The code executes the model on a given input request. This entry script receives data submitted to a deployed web service and passes it to the model, then returns the model's response to the client. The script is specific to your model. The entry script must understand the data that the model expects and returns. With an MLFlow model, as in this tutorial, this script is automatically created for you. Samples of scoring scripts can be found here.

Create a new online endpoint

Now that you have a registered model and an inference script, it's time to create your online endpoint. The endpoint name needs to be unique in the entire Azure region. For this tutorial, you'll create a unique name using [`UUID`] (https://en.wikipedia.org/wiki/Universally_unique_identifier#:~:text=A%20universally%20unique%20identifier%20(UUID,%2C%20for%20practical%20purposes%2C%20unique.).

```
import uuid

# Creating a unique name for the endpoint
online_endpoint_name = "credit-endpoint-" + str(uuid.uuid4())[:8]
```

[!NOTE] Expect the endpoint creation to take approximately 6 to 8 minutes.

```
from azure.ai.ml.entities import (
    ManagedOnlineEndpoint,
    ManagedOnlineDeployment,
    Model,
    Environment,
)

# create an online endpoint
endpoint = ManagedOnlineEndpoint(
    name=online_endpoint_name,
    description="this is an online endpoint",
    auth_mode="key",
    tags={
        "training_dataset": "credit_defaults",
        "model_type": "sklearn.GradientBoostingClassifier",
    },
)

endpoint =
ml_client.online_endpoints.begin_create_or_update(endpoint).result()

print(f"Endpoint {endpoint.name} provisioning state:
{endpoint.provisioning_state}")
```

Once you've created an endpoint, you can retrieve it as below:

```
endpoint = ml_client.online_endpoints.get(name=online_endpoint_name)

print(
    f'Endpoint "{endpoint.name}" with provisioning state
"{endpoint.provisioning_state}" is retrieved'
)
```

Deploy the model to the endpoint

Once the endpoint is created, deploy the model with the entry script. Each endpoint can have multiple deployments. Direct traffic to these deployments can be specified using rules. Here you'll create a single deployment that handles 100% of the incoming traffic. We have chosen a color name for the deployment, for example, *blue*, *green*, *red* deployments, which is arbitrary.

You can check the **Models** page on Azure ML studio, to identify the latest version of your registered model. Alternatively, the code below will retrieve the latest version number for you to use.

```
# Let's pick the latest version of the model
latest_model_version = max(
    [int(m.version) for m in
ml_client.models.list(name=registered_model_name)]
)
```

Deploy the latest version of the model.

[!NOTE] Expect this deployment to take approximately 6 to 8 minutes.

```
# picking the model to deploy. Here we use the latest version of our
registered model
model = ml_client.models.get(name=registered_model_name,
version=latest_model_version)


# create an online deployment.
blue_deployment = ManagedOnlineDeployment(
    name="blue",
    endpoint_name=online_endpoint_name,
    model=model,
    instance_type="Standard_DS3_v2",
    instance_count=1,
)

blue_deployment =
ml_client.begin_create_or_update(blue_deployment).result()
```

Test with a sample query

Now that the model is deployed to the endpoint, you can run inference with it.

Create a sample request file following the design expected in the run method in the score script.

```
deploy_dir = "./deploy"
os.makedirs(deploy_dir, exist_ok=True)

%%writefile {deploy_dir}/sample-request.json
{
  "input_data": {
    "columns":
[0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20,21,22],
    "index": [0, 1],
    "data": [
            [20000,2,2,1,24,2,2,-1,-1,-2,-
2,3913,3102,689,0,0,0,0,689,0,0,0,0],
            [10, 9, 8, 7, 6, 5, 4, 3, 2, 1, 10, 9, 8, 7, 6, 5, 4, 3,
```

```
2, 1, 10, 9, 8]
        ]
  }
}

# test the blue deployment with some sample data
ml_client.online_endpoints.invoke(
    endpoint_name=online_endpoint_name,
    request_file="./deploy/sample-request.json",
    deployment_name="blue",
)
```

Clean up resources

If you're not going to use the endpoint, delete it to stop using the resource. Make sure no other deployments are using an endpoint before you delete it.

> [!NOTE] Expect this step to take approximately 6 to 8 minutes.

```
ml_client.online_endpoints.begin_delete(name=online_endpoint_name)
```

Next Steps

Learn about creating a multi step pipeline for this script Create production ML pipelines in a Jupyter notebook.