Getting Started: training an image classification model

Learning Objectives - By the end of this quickstart tutorial, you'll know how to train and deploy an image classification model on Azure Machine Learning studio.

This tutorial covers:

- Connect to workspace & set up a compute resource on the Azure Machine Learning Studio Notebook UI
- Bring data in and prepare it to be used for training
- Train a model for image classification
- Metrics for optimizing your model
- Deploy the model online & test

1. Connect to Azure Machine Learning workspace

Before we dive in the code, you'll need to connect to your 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 are using `DefaultAzureCredential` to get access to workspace. `DefaultAzureCredential` should be capable of handling most scenarios. If you want to learn more about other available credentials, go to set up authentication doc, azure-identity reference doc.

Make sure to enter your workspace credentials before you run the script below.

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

# Authentication package
from azure.identity import DefaultAzureCredential

credential = DefaultAzureCredential()

# Get a handle to the workspace. You can find the info on the
workspace tab on ml.azure.com
ml_client = MLClient(
    credential=credential,
    subscription_id="<SUBSCRIPTION_ID>",  # this will look like
xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx
    resource_group_name="<RESOURCE_GROUP>",
    workspace_name="<AML_WORKSPACE_NAME>",
)
```

2. Create compute

In order to train a model on the Notebook editor on Azure Machine Learning studio, you will need to create a compute resource first. This is easily handled through a compute creation wizard. **Creating a compute will take 3-4 minutes.**

1. Click **...** menu button on the top of Notebook UI, and select **+Create Azure ML Compute Instance**.
2. **Name** the compute as **cpu-cluster**
3. Select **CPU** and **STANDARD_DS3_V2**.
4. Click **Create**

If you are interested in learning how to create compute via code, see Azure Machine Learning in a Day.

3. Create a job environment

To run an Azure Machine Learning training job, you'll need an environment.

In this tutorial, you'll using a ready-made environment called `AzureML-sklearn-1.0-ubuntu20.04-py38-cpu@latest` that contains all required libraries (python, MLflow, numpy, pip, etc).

4. Build the command job to train

Now that you have all assets required to run your job, it's time to build the job itself, using the Azure ML Python SDK v2. We will be creating a command job.

An AzureML command job is a resource that specifies all the details needed to execute your training code in the cloud: inputs and outputs, the type of hardware to use, software to install, and how to run your code. the command job contains information to execute a single command.

Create training script

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

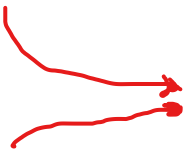
```
import os

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.

```
%%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_csv(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"),
    )
```



Model is saved and registered in the workspace.
Now, the registered model is used in the inferencing endpoints.

```
    # 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 running a script.

Here, you'll use input data, split ratio, learning rate and registered model name as input variables.

```
# import the libraries
from azure.ai.ml import command
from azure.ai.ml import Input

# name the model you registered earlier in the training script
registered_model_name = "credit_defaults_model"

# configure the command job
job = command(
    inputs=dict(
        # uri_file refers to a specific file as a data asset
        data=Input(
            type="uri_file",

path="https://azuremlexamples.blob.core.windows.net/datasets/credit_ca
rd/default%20of%20credit%20card%20clients.csv",
        ),
        test_train_ratio=0.2,  # input variable in main.py
        learning_rate=0.25,  # input variable in main.py
        registered_model_name=registered_model_name,  # input variable
in main.py
    ),
    code="./src/",  # location of source code
    # The inputs/outputs are accessible in the command via the ${{ ...
}} notation
    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}}",
    # This is the ready-made environment you are using
    environment="AzureML-sklearn-1.0-ubuntu20.04-py38-cpu@latest",
    # This is the compute you created earlier
```

```
    compute="cpu-cluster",
    # An experiment is a container for all the iterations one does on
a certain project. All the jobs submitted under the same experiment
name would be listed next to each other in Azure ML studio.
    experiment_name="train_model_credit_default_prediction",
    display_name="credit_default_prediction",
)
```

5. Submit the job

It's now time to submit the job to run in AzureML. **The job will take 2 to 3 minutes to run**. It could take longer (up to 10 minutes) if the compute instance has been scaled down to zero nodes and custom environment is still building.

```
# submit the command job
ml_client.create_or_update(job)
```

6. View the result of a training job

You can view the result of a training job by **clicking the URL generated after submitting a job**. Alternatively, you can also click **Jobs** on the left navigation menu. A job is a grouping of many runs from a specified script or piece of code. Information for the run is stored under that job.

1. **Overview** is where you can see the status of the job.

2. **Metrics** would display different visualizations of the metrics you specified in the script.
3. **Images** is where you can view any image artifacts that you have logged with MLflow.
4. **Child jobs** contains child jobs if you added them.
5. **Outputs + logs** contains log files you need for troubleshooting or other monitoring purposes.
6. **Code** contains the script/code used in the job.
7. **Explanations** and **Fairness** are used to see how your model performs against responsible AI standards. They are currently preview features and require additional package installations.
8. **Monitoring** is where you can view metrics for the performance of compute resources.

7. 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 via MLflow in *main.py*. You can find it under **Models** on the left navigation menu on Azure Machine Learning studio.
- The code that executes the model on a given input request. In this quickstart, you can easily set it up through the endpoint creation UI. If you want to learn more about how to deploy via Azure Machine Learning SDK, see Azure Machine Learning in a Day.

Find the endpoint creation wizard on Studio

1. Open a duplicate tab (so that you can keep this tutorial open).
2. On the duplicate tab, select **Endpoints** on the left navigation menu.
3. Select **+Create** for real-time endpoints.

Endpoint creation & deployment via wizard UI (this will take approximately 6 to 8 minutes)

1. Enter a **unique name** for *endpoint name*. We recommend creating a *unique* name with current date/time to avoid conflicts, which could prevent your deployment. Keep all the defaults for the rest.
2. Next, you need to choose a model to deploy. Select **credit_defaults_model** registered by *main.py* earlier.
3. Keep all the defaults for deployment configuration.
4. Select **Standard_DS3_V2** for compute, which is what we configured earlier. Set the instance count to **1**.
5. Keep all the defaults for the traffic.
6. Review: review and select **Create**.

Test with a sample query

1. Select the endpoint you just created. Make sure the endpoint is created and the model has been deployed to it.
2. Select the **Test** tab.
3. Copy & paste the following sample request file into the **Input data to test real-time endpoint** field.
4. Select **Test**.

```
{
  "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]
          ]
  }
}
```

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.

1. Click **Details** on the endpoint page.
2. Click the **Delete** button.

Expect this step to take approximately 6 to 8 minutes.