Notebook de prueba de Azure autoML

Adrián Robles Arques

January 8, 2025

This notebook was generated from the following AutoML run:

https://ml.azure.com/runs/test_vivienda_precio1_23?wsid=/subscriptions/61c99d29-0b57-45ee-85c3-cd01a43e41f8/resourcegroups/Data_Science/workspaces/Prueba1

1 Train using Azure Machine Learning Compute

- Connect to an Azure Machine Learning Workspace
- Use existing compute target or create new
- Configure & Run command

1.1 Prerequisites

Please ensure Azure Machine Learning Python SDK v2 is installed on the machine running Jupyter.

1.2 Connect to a Workspace

Initialize a workspace object from the previous experiment.

[ ]:
```
# Import the required libraries
from azure.identity import DefaultAzureCredential
from azure.ai.ml import MLClient

# The workspace information from the previous experiment has been pre-filled for␣
 ↪you.
subscription_id = "-"
resource_group = "-"
workspace_name = "Prueba1"

credential = DefaultAzureCredential()
ml_client = MLClient(credential, subscription_id, resource_group, workspace_name)
workspace = ml_client.workspaces.get(name=ml_client.workspace_name)
print(ml_client.workspace_name, workspace.resource_group, workspace.location,␣
 ↪ml_client.connections._subscription_id, sep = '\n')
```

1.2.1 Create project directory

Create a directory that will contain the training script that you will need access to on the remote resource.

[ ]:
```
import os
import shutil

project_folder = os.path.join(".", 'code_folder')
os.makedirs(project_folder, exist_ok=True)
shutil.copy('script.py', project_folder)
```

1.2.2 Use existing compute target or create new (Basic)

Azure Machine Learning Compute is managed compute infrastructure that allows the user to easily create single to multi-node compute of the appropriate VM Family. It is created **within your workspace region** and is a resource that can be used by other users in your workspace. It autoscales by default to the max_nodes, when a job is submitted, and executes in a containerized environment packaging the dependencies as specified by the user.

Since it is managed compute, job scheduling and cluster management are handled internally by Azure Machine Learning service.

A compute cluster can be created using the `AmlCompute` class. Some of the key parameters of this class are:

- `size` - The VM size to use for the cluster. For more information, see Supported VM series and sizes.
- `max_instances` - The maximum number of nodes to use on the cluster. Default is 1.

[ ]:
```
from azure.ai.ml.entities import AmlCompute

# Choose a name for your CPU cluster
cluster_name = "cpu-cluster"

# Verify that cluster does not exist already
try:
    cluster = ml_client.compute.get(cluster_name)
    print('Found existing cluster, use it.')
except Exception:
    compute = AmlCompute(name=cluster_name, size='STANDARD_DS4_V2',
                         max_instances=4)
    cluster = ml_client.compute.begin_create_or_update(compute)
```

1.2.3 Configure & Run

The environment and compute has been pre-filled from the original training job. More information can be found here:

`command`: https://docs.microsoft.com/en-us/python/api/azure-ai-ml/azure.ai.ml?view=azure-python-preview#azure-ai-ml-command

`environment`: https://docs.microsoft.com/en-us/azure/machine-learning/resource-curated-environments#automated-ml-automl

`compute`: https://docs.microsoft.com/en-us/python/api/azure-ai-ml/azure.ai.ml.entities.amlcompute?view=azure-python-preview

[ ]:
```
# To test the script with an environment referenced by a custom yaml file, uncomment the following lines and replace the `conda_file` value with the path to the yaml file.
# Set the value of `environment` in the `command` job below to `env`.

# env = Environment(
#     name="automl-tabular-env",
#     description="environment for automl inference",
#     image="mcr.microsoft.com/azureml/openmpi4.1.0-ubuntu20.04:20210727.v1",
#     conda_file="conda.yaml",
# )
```

[ ]:
```
from azure.ai.ml import command, Input

# To test with new training / validation datasets, replace the default dataset id(s)/uri(s) taken from parent run below
command_str = 'python script.py --training_dataset_uri azureml://locations/eastus2/workspaces/c5d90f81-e7a0-408d-b3a1-94f18e9cafb8/data/Datos_vivienda/versions/1'
command_job = command(
    code=project_folder,
    command=command_str,
    tags=dict(automl_child_run_id='test_vivienda_precio1_23'),
    environment='AzureML-ai-ml-automl:12',
    compute='cpu-cluster',
    experiment_name='Test1')

returned_job = ml_client.create_or_update(command_job)
returned_job.studio_url
```

1.2.4 Initialize MLFlow Client

The metrics and artifacts for the run can be accessed via the MLFlow interface. Initialize the MLFlow client here, and set the backend as Azure ML, via. the MLFlow Client.

IMPORTANT, you need to have installed the latest MLFlow packages with:

```
pip install azureml-mlflow

pip install mlflow
```

[ ]:
```
# %pip install azureml-mlflow
# %pip install mlflow
```

[ ]:
```
import mlflow

# Obtain the tracking URL from MLClient
MLFLOW_TRACKING_URI = ml_client.workspaces.get(
    name=ml_client.workspace_name
).mlflow_tracking_uri

# Set the MLFLOW TRACKING URI

mlflow.set_tracking_uri(MLFLOW_TRACKING_URI)

# Retrieve the metrics logged to the run.
from mlflow.tracking.client import MlflowClient

# Initialize MLFlow client
mlflow_client = MlflowClient()
mlflow_run = mlflow_client.get_run(returned_job.name)
mlflow_run.data.metrics
```

1.2.5 Download Fitted Model

Download the resulting fitted model to the local folder in `local_dir`.

[ ]:
```
# import os

# Create local folder
# local_dir = "./artifact_downloads"
# if not os.path.exists(local_dir):
#     os.mkdir(local_dir)
# Download run's artifacts/outputs
# local_path = mlflow_client.download_artifacts(
#     mlflow_run.info.run_id, "outputs", local_dir# )
# print("Artifacts downloaded in: {}".format(local_path))
# print("Artifacts: {}".format(os.listdir(local_path)))
```