CDW Documentation

Azure ML Pipeline Test

Azure ML Pipeline Test

Purpose

Test deployment of Azure ML Pipeline and look at outputs and download trained model.

Key Things Learned

1. When you create train.py and prep.py or any other environment files, they should be stored under ./src in your notebook directory.
2. You need to understand a bit of what you are trying to have it do as it can't think or make suppositions in a normal way. Garbage in, garbage out.

Final Code

train.py

```
import pandas as pd
import argparse
import os
from sklearn.linear_model import LogisticRegression
import joblib

def main():
    parser = argparse.ArgumentParser()
    parser.add_argument("--training_data", type=str)
    parser.add_argument("--model_output", type=str)
    args = parser.parse_args()

    df = pd.read_csv(os.path.join(args.training_data, "prepped.csv"))
    X = df[["feature1", "feature2", "feature_sum"]]
    y = df["label"]

    model = LogisticRegression()
    model.fit(X, y)

    os.makedirs(args.model_output, exist_ok=True)
    joblib.dump(model, os.path.join(args.model_output, "model.joblib"))

if __name__ == "__main__":
    main()
print("Model output path:", args.model_output)
print("Directory contents after writing:")
print(os.listdir(args.model_output))
print("Writing model to:", args.model_output)
```

```
print("Files in output dir:", os.listdir(args.model_output))
```

```
NOTE:  The print statements on the end were for troubleshooting and
shouldn't be there for production runs.
```

prep.py

```
import pandas as pd
import argparse
import os

def main():
    parser = argparse.ArgumentParser()
    parser.add_argument("--input_data", type=str)
    parser.add_argument("--output_data", type=str)
    args = parser.parse_args()

    df = pd.read_csv(args.input_data)
    df["feature_sum"] = df["feature1"] + df["feature2"]
    os.makedirs(args.output_data, exist_ok=True)
    df.to_csv(os.path.join(args.output_data, "prepped.csv"), index=False)

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

deployment_script.py

```
# Step 1: Install SDK
!pip install --quiet --upgrade azure-ai-ml

# Step 2: Imports and MLClient setup
from azure.ai.ml import MLClient, Input, Output, dsl
from azure.identity import DefaultAzureCredential
from azure.ai.ml.entities import Environment, CommandComponent, Data
from azure.ai.ml.constants import AssetTypes
import pandas as pd
import os
from uuid import uuid4

# Step 3: Connect to workspace
ml_client = MLClient(
    DefaultAzureCredential(),
    subscription_id="baa29726-b3e6-4910-bb9b-b585c655322c",
    resource_group_name="don-test-rg-SCUS",
    workspace_name="don-ml-workspace-fixed"
)

# Step 4: Create sample data and register it
df = pd.DataFrame({
    "feature1": [1, 2, 3, 4, 5],
    "feature2": [10, 20, 30, 40, 50],
```

```
    "label": [0, 1, 0, 1, 0]
})
df.to_csv("data.csv", index=False)

data_asset = Data(
    path="data.csv",
    type=AssetTypes.URI_FILE,
    description="Sample training data",
    name="sample-csv-data"
)
ml_client.data.create_or_update(data_asset)

# Step 5: Create Python scripts
os.makedirs("src", exist_ok=True)

# Leave train.py and prep.py creation to previous steps or user updates

# Step 6: Define Environment
env = Environment(
    name="basic-env",
    image="mcr.microsoft.com/azureml/openmpi4.1.0-ubuntu20.04:latest",
    conda_file={
        "name": "basic",
        "dependencies": [
            "python=3.8",
            "pandas",
            "scikit-learn",
            {
                "pip": [
                    "joblib"
                ]
            }
        ]
    }
)
ml_client.environments.create_or_update(env)

# Step 7: Create components from source
prep_component = CommandComponent(
    name="prep_data",
    description="Prep data component",
    inputs={"input_data": Input(type=AssetTypes.URI_FILE)},
    outputs={"output_data": Output(type=AssetTypes.URI_FOLDER)},
    code="./src",
    command="python prep.py --input_data ${{inputs.input_data}} --
output_data ${{outputs.output_data}}",
    environment=env,
    compute="cpu-cluster"
)
ml_client.components.create_or_update(prep_component)
```

```
# Force new train component to avoid cache issues
train_component = CommandComponent(
    name=f"train_model_{uuid4().hex[:8]}",  # unique name to bust cache
    description="Train model component",
    inputs={"training_data": Input(type=AssetTypes.URI_FOLDER)},
    outputs={"model_output": Output(type=AssetTypes.URI_FOLDER)},
    code="./src",
    command="python train.py --training_data ${{inputs.training_data}} --
model_output ${{outputs.model_output}}",
    environment=env,
    compute="cpu-cluster"
)
ml_client.components.create_or_update(train_component)

# Step 8: Define pipeline function
@dsl.pipeline(default_compute="cpu-cluster")
def ml_pipeline(input_data):
    prep_step = prep_component(input_data=input_data)
    train_step =
train_component(training_data=prep_step.outputs.output_data)
    return {"model_output": train_step.outputs.model_output}

# Step 9: Submit pipeline with explicit output registration
pipeline_job = ml_pipeline(
    input_data=Input(type=AssetTypes.URI_FILE, path="azureml:sample-csv-
data:4")
)

# Force Azure ML to track output
pipeline_job.outputs["model_output"] = Output(
    type=AssetTypes.URI_FOLDER,
    mode="rw_mount"
)

pipeline_job = ml_client.jobs.create_or_update(pipeline_job)

# Step 10: Stream logs
ml_client.jobs.stream(pipeline_job.name)
```

```
NOTE:  This is ran from the Notebook, not from a python script.  At least
not without changes.
```

AI Knowledge