CDW Documentation

Azure ML V2: Deploy and Query ML Model from CSV

Azure ML V2: Deploy and Query ML Model from CSV

Process

I exported my home MySQL database of collectibles to a csv file using:

```
mysql -u root -p -e "SELECT * FROM collectibles.inventory;" | sed
's/\t/","/g;s/^/"/;s/$/"/' > collectibles.csv
```

That gave me a file that looked like this:

```
"id","character","figure_name","property","type","manufacturer","location","
quantity","list_price","total_cost","approximate_value"
"1","Aang","Nendroid with Momo","Avatar the Last Airbender","Action
Figure","Good Smile Co","Collectible Room","1","40.00","40.00","53.20"
"2","Alice","D-Stage Alice","Alice In Wonderland","PVC Figure","Beast
Kingdom","Art Room","1","20.00","20.00","26.60"
"3","Alice","Disney Animators Collection Alice Doll","Alice In
Wonderland","Dolls","Disney","Art Room","1","29.99","29.99","39.89"
"4","Alice","Disney Store Classic","Alice In
Wonderland","Dolls","Disney","Art Room","1","19.99","19.99","26.59"
```

Then I used that file for the rest of the task.

There were many iterations to get to the working code and I noticed that the more exact you are, even when troubleshooting, the better off your are. I would give it the error messages and it would forget the things we already had done and would suggest it again. By telling it to forget everthing that happened before, it would wipe it's memory and start over.

Lessons Learned

The first step should be to know the steps involved or have AI tell you the steps before you ask it to do things. Had I started with asking what steps are required, I could have been more specific in my requests and probably gotten done more quickly.

Sometimes it's best to tell it to forget everything so it doesn't get in a loop.

Notebooks is a IDE that injects code into a compute instance or cluster and everything is additive until you restart the kernel. By this I mean if you define a script/variable that is saved to the compute instance, you can still access it from other scripts as you are “installing” things on the compute. When you restart the temporary memory is wiped and you have a fresh environment.

Final Working Code

Step 1: Upload CSV to Azure ML Notebooks

Upload your local 'collectibles.csv' to the Azure ML Notebooks file system using the file upload icon.

Purpose:
Upload and register the local `collectibles.csv` file into your Azure ML workspace as a named, versioned dataset so it can be used for training and analysis.

Step 2: Initialize MLClient

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

ml_client = MLClient(
  DefaultAzureCredential(),
  subscription_id="baa29726-b3e6-4910-bb9b-b585c655322c",
  resource_group_name="don-test-rg-SCUS",
  workspace_name="don-ml-workspace-fixed"
)
```

Purpose:
Authenticate and initialize the Azure ML client (`ml_client`) using your subscription, resource group, and workspace, enabling programmatic access to Azure ML resources.

Step 3: Load CSV and Train Model

```
import pandas as pd
from sklearn.ensemble import RandomForestRegressor
import joblib

df = pd.read_csv("collectibles.csv")
df = df.dropna(subset=["approximate_value"])
X = df[["character", "location"]].astype(str)
y = df["approximate_value"]

from sklearn.preprocessing import OneHotEncoder
from sklearn.compose import ColumnTransformer
from sklearn.pipeline import Pipeline

preprocessor = ColumnTransformer([
    ("cat", OneHotEncoder(handle_unknown="ignore"), ["character",
"location"])
])

model = Pipeline([
```

```
    ("preprocessor", preprocessor),
    ("regressor", RandomForestRegressor(n_estimators=10))
])

model.fit(X, y)
joblib.dump(model, "model.joblib")
```

Purpose:
Read the registered dataset into a Pandas DataFrame to validate its structure, inspect values, and ensure the data is suitable for training. Create and train a machine learning model based on the dataset, then serialize and register that model in Azure ML so it can be deployed later.

Step 4: Create score.py

```
# score.py
import joblib
import pandas as pd

def init():
    global model
    model = joblib.load("model.joblib")

def run(data):
    try:
        input_df = pd.DataFrame([{
            "character": data.get("character", ""),
            "location": data.get("location", "")
        }])
        prediction = model.predict(input_df)
        return {"prediction": prediction[0]}
    except Exception as e:
        return {"error": str(e)}
```

Purpose:
Develop a scoring script (`score.py`) that defines how the model will be loaded and how inference requests will be processed and responded to during deployment.

Step 5: Register Model

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

registered_model = ml_client.models.create_or_update(
    Model(
        path="model.joblib",
        name="collectibles-model",
        type="custom_model",
        description="Model predicting value based on character and location"
    )
```

```
)
```

Purpose:
Deploy the registered model to a managed online endpoint so it can serve inference requests in real time via REST API.

Step 6: Create and Deploy Endpoint

```
from azure.ai.ml.entities import ManagedOnlineEndpoint,
ManagedOnlineDeployment
import uuid

endpoint_name = f"collectibles-endpoint-{str(uuid.uuid4())[:8]}"

endpoint = ManagedOnlineEndpoint(
    name=endpoint_name,
    auth_mode="key"
)
ml_client.begin_create_or_update(endpoint).result()

deployment = ManagedOnlineDeployment(
    name="blue",
    endpoint_name=endpoint_name,
    model=registered_model.id,
    code_configuration={"code": ".", "scoring_script": "score.py"},
    instance_type="Standard_DS3_v2",
    instance_count=1
)
ml_client.begin_create_or_update(deployment).result()

ml_client.online_endpoints.begin_update(
    name=endpoint_name,
    update_parameters={"defaults": {"deployment_name": "blue"}}
).result()
```

Purpose:
Designate the deployed model (e.g., “blue” deployment) as the default handler for all requests sent to the online endpoint.

Step 7: Query the Endpoint

```
import requests
import json

endpoint = ml_client.online_endpoints.get(name=endpoint_name)
scoring_uri = endpoint.scoring_uri
key = ml_client.online_endpoints.get_keys(name=endpoint_name).primary_key
```

```
input_data = {
    "location": "Gotham",
    "character": "Batman"
}

headers = {
    "Content-Type": "application/json",
    "Authorization": f"Bearer {key}"
}

response = requests.post(scoring_uri, headers=headers,
data=json.dumps(input_data))
print("□ Status Code:", response.status_code)
print("□ Response:", response.text)
```

Purpose:
Send a test HTTP POST request with a query term (e.g., a character or location name) and verify the endpoint returns correct analytics: count and average value of matches.

AI Knowledge