

Highlight

Note

## Creating a Batch Inferencing Pipeline

To create a batch inferencing pipeline, perform the following tasks:

### 1. Register the Model

Just as with a real-time inferencing service, to use a trained model in a batch inferencing pipeline, you must register it in your Azure Machine Learning workspace.

To register a model from a local file, you can use the **register** method of the **Model** object as shown here:

```
from azureml.core import Model

classification_model = Model.register(workspace=ws,
                                     model_name='classification_model',
                                     model_path='model.pkl', # local path
                                     description='A classification model')
```

Alternatively, if you have a reference to the **Run** used to train the model, you can use its **register\_model** method as shown here:

```
run.register_model(model_name='classification_model',
                  model_path='outputs/model.pkl', # run outputs path
                  description='A classification model')
```

### 2. Create an Scoring Script

Just like a real-time inferencing service, a batch inferencing service requires a scoring script to load the model and use it to predict new values. It must include two functions:

- **init()**: Called when the pipeline is initialized.
- **run(mini\_batch)**: Called for each batch of data to be processed.

Typically, you use the **init** function to load the model from the model registry, and use the **run** function to generate predictions from each batch of data and return the results. The following example script shows this pattern:

```
import os
import numpy as np
from azureml.core import Model
import joblib

def init():
    # Runs when the pipeline step is initialized
    global model

    # load the model
    model_path = Model.get_model_path('classification_model')
    model = joblib.load(model_path)
```

```
def run(mini_batch):
    # This runs for each batch
    resultList = []

    # process each file in the batch
    for f in mini_batch:
        # Read comma-delimited data into an array
        data = np.genfromtxt(f, delimiter=',')
        # Reshape into a 2-dimensional array for model input
        prediction = model.predict(data.reshape(1, -1))
        # Append prediction to results
        resultList.append("{}: {}".format(os.path.basename(f), prediction[0]))
    return resultList
```

### 3. Create a Pipeline with a ParallelRunStep

Azure Machine Learning provides a type of pipeline step specifically for performing parallel batch inferencing. Using the **ParallelRunStep** class, you can read batches of files from a **File** dataset and write the processing output to a **PipelineData** reference. Additionally, you can set the **output\_action** setting for the step to “append\_row”, which will ensure that all instances of the step being run in parallel will collate their results to a single output file named *parallel\_run\_step.txt*:

```
from azureml.pipeline.steps import ParallelRunConfig, ParallelRunStep
from azureml.pipeline.core import PipelineData
from azureml.pipeline.core import Pipeline

# Get the batch dataset for input
batch_data_set = ws.datasets['batch-data']

# Set the output location
default_ds = ws.get_default_datastore()
output_dir = PipelineData(name='inferences',
                          datastore=default_ds,
                          output_path_on_compute='results')

# Define the parallel run step configuration
parallel_run_config = ParallelRunConfig(
    source_directory='batch_scripts',
    entry_script="batch_scoring_script.py",
    mini_batch_size="5",
    error_threshold=10,
    output_action="append_row",
    environment=batch_env,
    compute_target=aml_cluster,
    node_count=4)

# Create the parallel run step
parallelrun_step = ParallelRunStep(
    name='batch-score',
    parallel_run_config=parallel_run_config,
    inputs=[batch_data_set.as_named_input('batch_data')],
    output=output_dir,
    arguments=[],
    allow_reuse=True)
```

```
)  
# Create the pipeline  
pipeline = Pipeline(workspace=ws, steps=[parallelrun_step])
```

## 4. Run the pipeline and Retrieve the Step Output

After your pipeline has been defined, you can run it and wait for it to complete. Then you can retrieve the `parallel_run_step.txt` file from the output of the step to view the results.

```
from azureml.core import Experiment  
  
# Run the pipeline as an experiment  
pipeline_run = Experiment(ws, 'batch_prediction_pipeline').submit(pipeline)  
pipeline_run.wait_for_completion(show_output=True)  
  
# Get the outputs from the first (and only) step  
prediction_run = next(pipeline_run.get_children())  
prediction_output = prediction_run.get_output_data('inferences')  
prediction_output.download(local_path='results')  
  
# Find the parallel_run_step.txt file  
for root, dirs, files in os.walk('results'):  
    for file in files:  
        if file.endswith('parallel_run_step.txt'):  
            result_file = os.path.join(root, file)  
  
# Load and display the results  
df = pd.read_csv(result_file, delimiter=":", header=None)  
df.columns = ["File", "Prediction"]  
print(df)
```

This document belongs to Wolfgang Kiesenhofer.  
wolfgang.kiesenhofer@outlook.com  
No unauthorized copies allowed!

This document belongs to  
wolfgang.kiesenhofer@outlook.com  
No unauthorized copies allowed!