Azure Machine Learning Pipeline with AutoMLStep (Udacity Course 2)

This notebook demonstrates the use of AutoMLStep in Azure Machine Learning Pipeline.

Introduction

In this example we showcase how you can use AzureML Dataset to load data for AutoML via AML Pipeline.

If you are using an Azure Machine Learning Notebook VM, you are all set. Otherwise, make sure you have executed the configuration (https://aka.ms/pl-config) before running this notebook.

In this notebook you will learn how to:

1. Create an `Experiment` in an existing `Workspace`.
2. Create or Attach existing AmlCompute to a workspace.
3. Define data loading in a `TabularDataset`.
4. Configure AutoML using `AutoMLConfig`.
5. Use AutoMLStep
6. Train the model using AmlCompute
7. Explore the results.
8. Test the best fitted model.

Azure Machine Learning and Pipeline SDK-specific imports

In [1]:

```
import logging
import os
import csv

from matplotlib import pyplot as plt
import numpy as np
import pandas as pd
from sklearn import datasets
import pkg_resources

import azureml.core
from azureml.core.experiment import Experiment
```

```
from azureml.core.workspace import Workspace
from azureml.train.automl import AutoMLConfig
from azureml.core.dataset import Dataset

from azureml.pipeline.steps import AutoMLStep

# Check core SDK version number
print("SDK version:", azureml.core.VERSION)
```

```
SDK version: 1.19.0
```

Initialize Workspace

Initialize a workspace object from persisted configuration. Make sure the config file is present at .\config.json

In [2]:

```
ws = Workspace.from_config()
print(ws.name, ws.resource_group, ws.location, ws.s
ubscription_id, sep = '\n')
```

```
Performing interactive authentication. Please follo
w the instructions on the terminal.
To sign in, use a web browser to open the page http
s://microsoft.com/devicelogin and enter the code FX
FQEDTEH to authenticate.
You have logged in. Now let us find all the subscri
ptions to which you have access...
Interactive authentication successfully completed.
quick-starts-ws-130929
aml-quickstarts-130929
southcentralus
30d182b7-c8c4-421c-8fa0-d3037ecfe6d2
```

Create an Azure ML experiment

Let's create an experiment named "automlstep-classification" and a folder to hold the training scripts. The script runs will be recorded under the experiment in Azure.

The best practice is to use separate folders for scripts and its dependent files for each step and specify that folder as the `source_directory` for the step. This helps reduce the size of the snapshot created for the step (only the specific folder is snapshotted). Since changes in any files in the `source_directory` would trigger a re-upload of the snapshot, this helps keep the reuse of the step when there are no changes in the `source_directory` of the step.

Udacity Note: There is no need to create an Azure ML experiment, this needs to re-use the experiment that was already created

In [3]:

In [3]:

```
# Choose a name for the run history container in th
e workspace.
# NOTE: update these to match your existing experim
ent name
experiment_name = 'udacity-ml-project-2'
project_folder = './pipeline-project'

experiment = Experiment(ws, experiment_name)
experiment
```

Out[3]:

Name	Workspace	Report Page
udacity-ml-project-2	quick-starts-ws-130929	Link to Azure Machine Learning studio (https://ml.azure.com/experiments/udacity-ml-project-2?wsid=/subscriptions/30d182b7-c8c4-421c-8fa0-d3037ecfe6d2/resourcegroups/aml-quickstarts-130929/workspaces/quick-starts-ws-130929)

Create or Attach an AmlCompute cluster

You will need to create a compute target (https://docs.microsoft.com/azure/machine-learning/service/concept-azure-machine-learning-architecture#compute-target) for your AutoML run. In this tutorial, you get the default `AmlCompute` as your training compute resource.

Udacity Note There is no need to create a new compute target, it can re-use the previous cluster

In [4]:

```
from azureml.core.compute import AmlCompute
from azureml.core.compute import ComputeTarget
from azureml.core.compute_target import ComputeTarg
etException

# NOTE: update the cluster name to match the existi
ng cluster
# Choose a name for your CPU cluster
amlcompute_cluster_name = "udacity-banking"

# Verify that cluster does not exist already
try:
    compute_target = ComputeTarget(workspace=ws, na
me=amlcompute_cluster_name)
```

```
    print('Found existing cluster, use it.')
except ComputeTargetException:
    compute_config = AmlCompute.provisioning_config
uration(vm_size='STANDARD_D2_V2',# for GPU, use "ST
ANDARD_NC6"

#vm_priority = 'lowpriority', # optional

max_nodes=4)
    compute_target = ComputeTarget.create(ws, amlco
mpute_cluster_name, compute_config)

compute_target.wait_for_completion(show_output=True
, min_node_count = 1, timeout_in_minutes = 10)
# For a more detailed view of current AmlCompute st
atus, use get_status().
```

```
Creating
Succeede
d...........................................................................

AmlCompute wait for completion finished

Wait timeout has been reached
Current provisioning state of AmlCompute is "Succee
ded" and current node count is "0"
```

Data

Udacity note: Make sure the `key` is the same name as the dataset that is uploaded, and that the description matches. If it is hard to find or unknown, loop over the `ws.datasets.keys()` and `print()` them. If it *isn't* found because it was deleted, it can be recreated with the link that has the CSV

In [5]:

```
# Try to load the dataset from the Workspace. Other
wise, create it from the file
# NOTE: update the key to match the dataset name
found = False
key = "BankMarketing Dataset"
description_text = "Bank Marketing DataSet for Udac
ity Course 2"

if key in ws.datasets.keys():
        found = True
        dataset = ws.datasets[key]

if not found:
        # Create AML Dataset and register it into W
orkspace
        example_data = 'https://automlsamplenoteboo
kdata.blob.core.windows.net/automl-sample-notebook-
data/bankmarketing_train.csv'
        dataset = Dataset.Tabular.from_delimited_fi
```

```
les(example_data)
        #Register Dataset in Workspace
        dataset = dataset.register(workspace=ws,
                                   name=key,
                                   description=desc
ription_text)


df = dataset.to_pandas_dataframe()
df.describe()
```

Out[5]:

	age	duration	campaign	pdays
count	32950.000000	32950.000000	32950.000000	32950.00
mean	40.040212	257.335205	2.561730	962.1747
std	10.432313	257.331700	2.763646	187.6467
min	17.000000	0.000000	1.000000	0.000000
25%	32.000000	102.000000	1.000000	999.0000
50%	38.000000	179.000000	2.000000	999.0000
75%	47.000000	318.000000	3.000000	999.0000
max	98.000000	4918.000000	56.000000	999.0000

Review the Dataset Result

You can peek the result of a TabularDataset at any range using `skip(i)` and `take(j).to_pandas_dataframe()`. Doing so evaluates only `j` records for all the steps in the TabularDataset, which makes it fast even against large datasets.

`TabularDataset` objects are composed of a list of transformation steps (optional).

In [6]:

```
dataset.take(5).to_pandas_dataframe()
```

Out[6]:

	age	job	marital	education	default	housing	lo
0	57	technician	married	high.school	no	no	ye
1	55	unknown	married	unknown	unknown	yes	nc
2	33	blue-collar	married	basic.9y	no	no	nc
3	36	admin.	married	high.school	no	no	nc
4	27	housemaid	married	high.school	no	yes	nc

5 rows × 21 columns

Train

This creates a general AutoML settings object. **Udacity notes:** These inputs must match what was used when training in the portal. `label_column_name` has to be y for example.

In [7]:

```
automl_settings = {
    "experiment_timeout_minutes": 20,
    "max_concurrent_iterations": 5,
    "primary_metric" : 'AUC_weighted'
}
automl_config = AutoMLConfig(compute_target=compute
_target,
                             task = "classificatio
n",
                             training_data=dataset,
                             label_column_name="y",
                             path = project_folder,
                             enable_early_stopping=
True,
                             featurization= 'auto',
                             debug_log = "automl_er
rors.log",
                             **automl_settings
                            )
```

Create Pipeline and AutoMLStep

You can define outputs for the AutoMLStep using TrainingOutput.

In [8]:

```
from azureml.pipeline.core import PipelineData, Tra
iningOutput

ds = ws.get_default_datastore()
metrics_output_name = 'metrics_output'
best_model_output_name = 'best_model_output'

metrics_data = PipelineData(name='metrics_data',
                            datastore=ds,
                            pipeline_output_name=met
rics_output_name,
                            training_output=Training
Output(type='Metrics'))
model_data = PipelineData(name='model_data',
                            datastore=ds,
                            pipeline_output_name=bes
t_model_output_name,
                            training_output=Training
```

```
Output(type='Model'))
```

Create an AutoMLStep.

In [9]:

```
automl_step = AutoMLStep(
    name='automl_module',
    automl_config=automl_config,
    outputs=[metrics_data, model_data],
    allow_reuse=True)
```

In [10]:

```
from azureml.pipeline.core import Pipeline
pipeline = Pipeline(
    description="pipeline_with_automlstep",
    workspace=ws,
    steps=[automl_step])
```

In [11]:

```
pipeline_run = experiment.submit(pipeline)
```

```
Created step automl_module [47fb6778][3ef0ba4d-bdc9
-4b5c-9859-e2ec110f3352], (This step will run and g
enerate new outputs)
Submitted PipelineRun 3c0b0ad0-138d-4f52-b6a8-e74d9
8462c20
Link to Azure Machine Learning Portal: https://ml.a
zure.com/experiments/udacity-ml-project-2/runs/3c0b
0ad0-138d-4f52-b6a8-e74d98462c20?wsid=/subscription
s/30d182b7-c8c4-421c-8fa0-d3037ecfe6d2/resourcegrou
ps/aml-quickstarts-130929/workspaces/quick-starts-w
s-130929
```

In [12]:

```
from azureml.widgets import RunDetails
RunDetails(pipeline_run).show()
```

```
_PipelineWidget(widget_settings={'childWidgetDispla
y': 'popup', 'send_telemetry': False, 'log_level':
'INFO', …
```

In [13]:

```
pipeline_run.wait_for_completion()
```

```
PipelineRunId: 3c0b0ad0-138d-4f52-b6a8-e74d98462c20
Link to Azure Machine Learning Portal: https://ml.a
zure.com/experiments/udacity-ml-project-2/runs/3c0b
0ad0-138d-4f52-b6a8-e74d98462c20?wsid=/subscription
s/30d182b7-c8c4-421c-8fa0-d3037ecfe6d2/resourcegrou
ps/aml-quickstarts-130929/workspaces/quick-starts-w
s-130929
PipelineRun Status: Running
```

```
PipelineRun Execution Summary
==============================
PipelineRun Status: Finished
{'runId': '3c0b0ad0-138d-4f52-b6a8-e74d98462c20',
'status': 'Completed', 'startTimeUtc': '2020-12-18T
21:09:48.96795Z', 'endTimeUtc': '2020-12-18T21:49:1
0.214845Z', 'properties': {'azureml.runsource': 'az
ureml.PipelineRun', 'runSource': 'SDK', 'runType':
'SDK', 'azureml.parameters': '{}'}, 'inputDataset
s': [], 'logFiles': {'logs/azureml/executionlogs.tx
t': 'https://mlstrg130929.blob.core.windows.net/azu
reml/ExperimentRun/dcid.3c0b0ad0-138d-4f52-b6a8-e74
d98462c20/logs/azureml/executionlogs.txt?sv=2019-02
-02&sr=b&sig=TuXnZv9AEn6%2BHvHVKt3YCMOlXpd9AQSnucuE
ZVNfwZ8%3D&st=2020-12-18T21%3A00%3A13Z&se=2020-12-1
9T05%3A10%3A13Z&sp=r', 'logs/azureml/stderrlogs.tx
t': 'https://mlstrg130929.blob.core.windows.net/azu
reml/ExperimentRun/dcid.3c0b0ad0-138d-4f52-b6a8-e74
d98462c20/logs/azureml/stderrlogs.txt?sv=2019-02-02
&sr=b&sig=7WzovPWfTpMiE3vn%2BNojluyLgEGtBCcDrPJf6kP
LbZ8%3D&st=2020-12-18T21%3A00%3A13Z&se=2020-12-19T0
5%3A10%3A13Z&sp=r', 'logs/azureml/stdoutlogs.txt':
'https://mlstrg130929.blob.core.windows.net/azurem
l/ExperimentRun/dcid.3c0b0ad0-138d-4f52-b6a8-e74d98
462c20/logs/azureml/stdoutlogs.txt?sv=2019-02-02&sr
=b&sig=jKM6XJ4uG%2BCvJeQl1NsEFFGa2dh1jN1mniAU5U%2BP
tow%3D&st=2020-12-18T21%3A00%3A13Z&se=2020-12-19T0
5%3A10%3A13Z&sp=r'}}
```

```
WARNING:azureml.pipeline.core.run:Expected a StepRu
n object but received <class 'azureml.core.run.Ru
n'> instead.
This usually indicates a package conflict with one
of the dependencies of azureml-core or azureml-pipe
line-core.
Please check for package conflicts in your python e
nvironment
```

Out[13]:

```
'Finished'
```

Examine Results

Retrieve the metrics of all child runs

Outputs of above run can be used as inputs of other steps in pipeline. In this tutorial, we will examine the outputs by retrieve output data and running some tests.

In [14]:

```
metrics_output = pipeline_run.get_pipeline_output(m
```

```
etrics_output_name)
num_file_downloaded = metrics_output.download('.', show_progress=True)
```

```
Downloading azureml/623c8676-d08a-4e02-9ff2-df14fc3337bd/metrics_data
Downloaded azureml/623c8676-d08a-4e02-9ff2-df14fc3337bd/metrics_data, 1 files out of an estimated total of 1
```

In [15]:

```
import json
with open(metrics_output._path_on_datastore) as f:
    metrics_output_result = f.read()

deserialized_metrics_output = json.loads(metrics_output_result)
df = pd.DataFrame(deserialized_metrics_output)
df
```

Out[15]:

	623c8676-d08a-4e02-9ff2-df14fc3337bd_9
f1_score_micro	[0.7226100151745067]
f1_score_weighted	[0.7716429917171808]
AUC_micro	[0.8187225321853824]
f1_score_macro	[0.5975078236343788]
AUC_macro	[0.8293127497235329]
norm_macro_recall	[0.4579065920529335]
average_precision_score_macro	[0.7082843042787298]
average_precision_score_micro	[0.7852735772887296]
log_loss	[0.5570028527866079]
recall_score_weighted	[0.7226100151745068]
recall_score_macro	[0.7289532960264667]
precision_score_macro	[0.6028997216873258]
weighted_accuracy	[0.721035150121809]
balanced_accuracy	[0.7289532960264667]
matthews_correlation	[0.3069803279724571]
accuracy	[0.7226100151745068]
precision_score_weighted	[0.8769366245044448]
AUC_weighted	[0.8293127497235329]
average_precision_score_weighted	[0.9128484421266932]

recall_score_micro	[0.7226100151745068]
precision_score_micro	[0.7226100151745068]

21 rows × 36 columns

Retrieve the Best Model

In [16]:

```
# Retrieve best model from Pipeline Run
best_model_output = pipeline_run.get_pipeline_outpu
t(best_model_output_name)
num_file_downloaded = best_model_output.download(
'.', show_progress=True)
```

```
Downloading azureml/623c8676-d08a-4e02-9ff2-df14fc3
337bd/model_data
Downloaded azureml/623c8676-d08a-4e02-9ff2-df14fc33
37bd/model_data, 1 files out of an estimated total
of 1
```

In [17]:

```
import pickle

with open(best_model_output._path_on_datastore, "r
b" ) as f:
    best_model = pickle.load(f)
best_model
```

Out[17]:

```
PipelineWithYTransformations(Pipeline={'memory': No
ne,
                                         'steps':
[('datatransformer',
                                                    D
ataTransformer(enable_dnn=None,

enable_feature_sweeping=None,

feature_sweeping_config=None,

feature_sweeping_timeout=None,

featurization_config=None,

force_text_dnn=None,

is_cross_validation=None,

is_onnx_compatible=None,

logger=None,

observer=None,
```

```
task=None,

working_dir=None))...

n_jobs=1,

oob_score=False,

random_state=None,

verbose=0,

warm_start=False))],

verbose=False))],

flatten_transform=None,

weights=[0.3333333333333333,

0.26666666666666666,

0.06666666666666667,

0.06666666666666667,

0.06666666666666667,

0.06666666666666667,

0.06666666666666667,

0.06666666666666667]))],
                                               'verbose': F
alse},
                             y_transformer={},
                             y_transformer_name='La
belEncoder')
```

In [18]:

```
best_model.steps
```

Out[18]:

```
[('datatransformer',
  DataTransformer(enable_dnn=None, enable_feature_s
weeping=None,
                  feature_sweeping_config=None, fea
ture_sweeping_timeout=None,
                  featurization_config=None, force_
text_dnn=None,
                  is_cross_validation=None, is_onnx
_compatible=None, logger=None,
                  observer=None, task=None, working
_dir=None)),
 ('prefittedsoftvotingclassifier',
```

```
  PreFittedSoftVotingClassifier(classification_labe
ls=None,
                                estimators=[('0',
                                             Pipeli
ne(memory=None,

steps=[('maxabsscaler',
```