Train a classification model with Automated Machine Learning

There are many kinds of machine learning algorithm that you can use to train a model, and sometimes it's not easy to determine the most effective algorithm for your particular data and prediction requirements. Additionally, you can significantly affect the predictive performance of a model by preprocessing the training data, using techniques such as normalization, missing feature imputation, and others. In your quest to find the best model for your requirements, you may need to try many combinations of algorithms and preprocessing transformations; which takes a lot of time and compute resources.

Azure Machine Learning enables you to automate the comparison of models trained using different algorithms and preprocessing options. You can use the visual interface in Azure Machine Learning Studio or the Python SDK (v2) to leverage this capability. The Python SDK gives you greater control over the settings for the automated machine learning job, but the visual interface is easier to use.

Before you start

You'll need the latest version of the azureml-ai-ml package to run the code in this notebook. Run the cell below to verify that it is installed.

> **Note: If the azure-ai-ml package is not installed, run `pip install azure-ai-ml` to install it.**

In [ ]:

```
pip show azure-ai-ml
```

Connect to your workspace

With the required SDK packages installed, now you're ready to connect to your workspace.

To connect to a workspace, we need identifier parameters - a subscription ID, resource group name, and workspace name. Since you're working with a compute instance, managed by Azure Machine Learning, you can use the default values to connect to the workspace.

In [ ]:

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

try:
    credential = DefaultAzureCredential()
    # Check if given credential can get token successfully.
    credential.get_token("https://management.azure.com/.default")
except Exception as ex:
    # Fall back to InteractiveBrowserCredential in case DefaultAzureCredential not work
    credential = InteractiveBrowserCredential()
```

In [ ]:

```
# Get a handle to workspace
ml_client = MLClient.from_config(credential=credential)
```

Prepare data

You don't need to create a training script for automated machine learning, but you do need to load the training data.

In this case, you'll use a dataset containing details of diabetes patients.

To pass a dataset as an input to an automated machine learning job, the data must be in tabular form and include a target column. For the data to be interpreted as a tabular dataset, the input dataset must be a MLTable.

A MLTable data asset has already been created for you during set-up. You can explore the data asset by navigating to the Data page. You'll retrieve the data asset here by specifying its name `diabetes-training-table` and version `1`.

In [ ]:

```
from azure.ai.ml.constants import AssetTypes
from azure.ai.ml import Input

# creates a dataset based on the files in the local data folder
my_training_data_input = Input(type=AssetTypes.MLTABLE, path="azureml:diabetes-training:1
")
```

Configure automated machine learning job

Now, you're ready to configure the automated machine learning experiment.

When you run the code below, it will create an automated machine learning job that:

- **Uses the compute cluster named `aml-cluster`**
- **Sets `Diabetic` as the target column**
- **Sets `accuracy` as the primary metric**
- **Times out after `60` minutes of total training time**
- **Trains a maximum of `5` models**
- **No model will be trained with the `LogisticRegression` algorithm**

In [ ]:

```
from azure.ai.ml import automl

# configure the classification job
classification_job = automl.classification(
    compute="aml-cluster",
    experiment_name="auto-ml-class-dev",
    training_data=my_training_data_input, (Input dataset must be a MLTable)
    target_column_name="Diabetic",
    primary_metric="accuracy",
    n_cross_validations=5,
    enable_model_explainability=True
)

# set the limits (optional)
classification_job.set_limits(
    timeout_minutes=60,
    trial_timeout_minutes=20,
    max_trials=5,
    enable_early_termination=True,
)

# set the training properties (optional)
classification_job.set_training(
    blocked_training_algorithms=["LogisticRegression"],
    enable_onnx_compatible_models=True
)
```

Run an automated machine learning job

OK, you're ready to go. Let's run the automated machine learning experiment.

> **Note:** This may take some time!

In [ ]:

```
# Submit the AutoML job
returned_job = ml_client.jobs.create_or_update(
    classification_job
)

# submit the job to the backend
aml_url = returned_job.studio_url
print("Monitor your job at", aml_url)
```

While the job is running, you can monitor it in the Studio.