Run a training script as a command job

You can use the Python SDK for Azure Machine Learning to submit scripts as command jobs. By using jobs, you can easily keep track of the input parameters and outputs when training a machine learning model.

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)
```

Custom tracking with MLflow

When running a script as a job you can use MLflow in your training script to track the model. MLflow allows you to track any custom parameters, metrics, or artifacts you want to store with your job output.

Run the following cells to create the train-model-mlflow.py script in the src folder. The script trains a classification model by using the diabetes.csv file in the same folder, which is passed as an argument.

Review the code below to find that the script will import `mlflow` and log:

- **The regularization rate as a parameter.**
- **The accuracy and AUC as metrics.**
- **The plotted ROC curve as an artifact.**

In [ ]:

```
import os

# create a folder for the script files
script_folder = 'src'
os.makedirs(script_folder, exist_ok=True)
print(script_folder, 'folder created')
```

In [ ]:

```
%%writefile $script_folder/train-model-mlflow.py
# import libraries
import mlflow
import argparse
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import roc_auc_score
from sklearn.metrics import roc_curve
import matplotlib.pyplot as plt

def main(args):
    # read data
    df = get_data(args.training_data)

    # split data
    X_train, X_test, y_train, y_test = split_data(df)

    # train model
    model = train_model(args.reg_rate, X_train, X_test, y_train, y_test)

    # evaluate model
    eval_model(model, X_test, y_test)

# function that reads the data
def get_data(path):
    print("Reading data...")
    df = pd.read_csv(path)

    return df

# function that splits the data
def split_data(df):
    print("Splitting data...")
    X, y = df[['Pregnancies','PlasmaGlucose','DiastolicBloodPressure','TricepsThickness'
,
    'SerumInsulin','BMI','DiabetesPedigree','Age']].values, df['Diabetic'].values

    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.30, random_sta
te=0)

    return X_train, X_test, y_train, y_test

# function that trains the model
def train_model(reg_rate, X_train, X_test, y_train, y_test):
    mlflow.log_param("Regularization rate", reg_rate)
    print("Training model...")
    model = LogisticRegression(C=1/reg_rate, solver="liblinear").fit(X_train, y_train)

    return model

# function that evaluates the model
def eval_model(model, X_test, y_test):
    # calculate accuracy
    y_hat = model.predict(X_test)
    acc = np.average(y_hat == y_test)
    print('Accuracy:', acc)
    mlflow.log_metric("Accuracy", acc)

    # calculate AUC
    y_scores = model.predict_proba(X_test)
```

```
    auc = roc_auc_score(y_test,y_scores[:,1])
    print('AUC: ' + str(auc))
    mlflow.log_metric("AUC", auc)

    # plot ROC curve
    fpr, tpr, thresholds = roc_curve(y_test, y_scores[:,1])
    fig = plt.figure(figsize=(6, 4))
    # Plot the diagonal 50% line
    plt.plot([0, 1], [0, 1], 'k--')
    # Plot the FPR and TPR achieved by our model
    plt.plot(fpr, tpr)
    plt.xlabel('False Positive Rate')
    plt.ylabel('True Positive Rate')
    plt.title('ROC Curve')
    plt.savefig("ROC-Curve.png")
    mlflow.log_artifact("ROC-Curve.png")

def parse_args():
    # setup arg parser
    parser = argparse.ArgumentParser()

    # add arguments
    parser.add_argument("--training_data", dest='training_data',
                        type=str)
    parser.add_argument("--reg_rate", dest='reg_rate',
                        type=float, default=0.01)

    # parse args
    args = parser.parse_args()

    # return args
    return args

# run script
if __name__ == "__main__":
    # add space in logs
    print("\n\n")
    print("*" * 60)

    # parse args
    args = parse_args()

    # run main function
    main(args)

    # add space in logs
    print("*" * 60)
    print("\n\n")
```

Now, you can submit the script as a command job.

Run the cell below to train the model.

In [ ]:

```
from azure.ai.ml import command

# configure job

job = command(
    code="./src",
    command="python train-model-mlflow.py --training_data diabetes.csv",
    environment="AzureML-sklearn-0.24-ubuntu18.04-py37-cpu@latest",
    compute="aml-cluster",
    display_name="diabetes-train-mlflow",
    experiment_name="diabetes-training",
    tags={"model_type": "LogisticRegression"}
    )

# submit job
returned_job = ml_client.create_or_update(job)
```

```
aml_url = returned_job.studio_url
print("Monitor your job at", aml_url)
```

In the Studio, navigate to the diabetes-train-mlflow job to explore the overview of the command job you ran:

- **Find the logged parameters in the Overview tab, under Params.**
- **Find the logged metrics in the Metrics tab.**
- **Find the logged artifacts in the Images tab (specifically for images), and in the Outputs + logs tab (all files).**

Autologging with MLflow

Instead of using custom logging, MLflow can also automatically log any parameters, metrics, and artifacts. Autologging with MLflow requires only one line of code.

Run the following cell to create the train-model-autolog.py script in the src folder. The script trains a classification model by using the diabetes.csv file in the same folder, which is passed as an argument.

Review the code below to find that the script will import `mlflow` and enables autologging with the line:

```
mlflow.autolog()
```

In [ ]:

```
%%writefile $script_folder/train-model-autolog.py
# import libraries
import mlflow
import argparse
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import roc_auc_score
from sklearn.metrics import roc_curve
import matplotlib.pyplot as plt

def main(args):
    # enable autologging
    mlflow.autolog()

    # read data
    df = get_data(args.training_data)

    # split data
    X_train, X_test, y_train, y_test = split_data(df)

    # train model
    model = train_model(args.reg_rate, X_train, X_test, y_train, y_test)

    eval_model(model, X_test, y_test)

# function that reads the data
def get_data(path):
    print("Reading data...")
    df = pd.read_csv(path)

    return df

# function that splits the data
def split_data(df):
    print("Splitting data...")
    X, y = df[['Pregnancies','PlasmaGlucose','DiastolicBloodPressure','TricepsThickness'
,
    'SerumInsulin','BMI','DiabetesPedigree','Age']].values, df['Diabetic'].values

    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.30, random_sta
te=0)
```

```
    return X_train, X_test, y_train, y_test

# function that trains the model
def train_model(reg_rate, X_train, X_test, y_train, y_test):
    mlflow.log_param("Regularization rate", reg_rate)
    print("Training model...")
    model = LogisticRegression(C=1/reg_rate, solver="liblinear").fit(X_train, y_train)

    return model

# function that evaluates the model
def eval_model(model, X_test, y_test):
    # calculate accuracy
    y_hat = model.predict(X_test)
    acc = np.average(y_hat == y_test)
    print('Accuracy:', acc)

    # calculate AUC
    y_scores = model.predict_proba(X_test)
    auc = roc_auc_score(y_test,y_scores[:,1])
    print('AUC: ' + str(auc))

    # plot ROC curve
    fpr, tpr, thresholds = roc_curve(y_test, y_scores[:,1])
    fig = plt.figure(figsize=(6, 4))
    # Plot the diagonal 50% line
    plt.plot([0, 1], [0, 1], 'k--')
    # Plot the FPR and TPR achieved by our model
    plt.plot(fpr, tpr)
    plt.xlabel('False Positive Rate')
    plt.ylabel('True Positive Rate')
    plt.title('ROC Curve')
    plt.savefig("ROC-Curve.png")

def parse_args():
    # setup arg parser
    parser = argparse.ArgumentParser()

    # add arguments
    parser.add_argument("--training_data", dest='training_data',
                        type=str)
    parser.add_argument("--reg_rate", dest='reg_rate',
                        type=float, default=0.01)

    # parse args
    args = parser.parse_args()

    # return args
    return args

# run script
if __name__ == "__main__":
    # add space in logs
    print("\n\n")
    print("*" * 60)

    # parse args
    args = parse_args()

    # run main function
    main(args)

    # add space in logs
    print("*" * 60)
    print("\n\n")
```

Now, you can submit the script as a command job.

Run the cell below to train the model.

In [ ]:

```
from azure.ai.ml import command

# configure job

job = command(
    code="./src",
    command="python train-model-autolog.py --training_data diabetes.csv",
    environment="AzureML-sklearn-0.24-ubuntu18.04-py37-cpu@latest",
    compute="aml-cluster",
    display_name="diabetes-train-autolog",
    experiment_name="diabetes-training"
    )

# submit job
returned_job = ml_client.create_or_update(job)
aml_url = returned_job.studio_url
print("Monitor your job at", aml_url)
```

In the Studio, navigate to the diabetes-train-autolog job to explore the overview of the command job you ran:

- **Find the logged parameters in the Overview tab, under Params.**
- **Find the logged metrics in the Metrics tab.**
- **Find the logged artifacts in the Images tab (specifically for images), and in the Outputs + logs tab (all files, including the model files).**

Use MLflow to view and search for experiments

The Azure Machine Learning Studio is an easy-to-use UI to view and compare job runs. Alternatively, you can use MLflow to view experiment jobs.

To list the jobs in the workspace, use the following command to list the experiments in the workspace:

In [ ]:

```
import mlflow
experiments = mlflow.search_experiments()
for exp in experiments:
    print(exp.name)
```

To retrieve a specific experiment, you can get it by its name:

In [ ]:

```
experiment_name = "diabetes-training"
exp = mlflow.get_experiment_by_name(experiment_name)
print(exp)
```

Using an experiment name, you can retrieve all jobs of that experiment:

In [ ]:

```
mlflow.search_runs(exp.experiment_id)
```

To more easily compare job runs and outputs, you can configure the search to order the results. For example, the following cell orders the results by `start_time`, and only shows a maximum of `2` results:

In [ ]:

```
mlflow.search_runs(exp.experiment_id, order_by=["start_time DESC"], max_results=2)
```

You can even create a query to filter the runs. Filter query strings are written with a simplified version of the SQL `WHERE` clause.

To filter, you can use two classes of comparators:

To filter, you can use two classes of comparators:

- **Numeric comparators (metrics): =, !=, >, >=, <, and <=.**
- **String comparators (params, tags, and attributes): = and !=.**

Learn more about [how to track experiments with MLflow](#) .

In [ ]:

```
query = "metrics.AUC > 0.8 and tags.model_type = 'LogisticRegression'"
mlflow.search_runs(exp.experiment_id, filter_string=query)
```