Tutorial: Train your first model

This tutorial is **part two of a two-part tutorial series**. In the previous tutorial, you created a workspace and chose a development environment. In this tutorial, you learn the foundational design patterns in Azure Machine Learning service, and train a simple scikit-learn model based on the diabetes data set. After completing this tutorial, you will have the practical knowledge of the SDK to scale up to developing more-complex experiments and workflows.

In this tutorial, you learn the following tasks:

> - Connect your workspace and create an experiment
> - Load data and train a scikit-learn model
> - View training results in the studio
> - Retrieve the best model

Prerequisites

The only prerequisite is to run the previous tutorial, Setup environment and workspace.

Connect workspace and create experiment

Import the `Workspace` class, and load your subscription information from the file `config.json` using the function `from_config()`. This looks for the JSON file in the current directory by default, but you can also specify a path parameter to point to the file using `from_config(path="your/file/path")` . If you are running this notebook in a cloud notebook server in your workspace, the file is automatically in the root directory.

If the following code asks for additional authentication, simply paste the link in a browser and enter the authentication token. In addition, if you have more than one tenant linked to your user, you will need to add the following lines:

```
from azureml.core.authentication import
InteractiveLoginAuthentication
interactive_auth =
InteractiveLoginAuthentication(tenant_id="your-tenant-id")
Additional details on authentication can be found here:
https://aka.ms/aml-notebook-auth
```

In [ ]:

```
from azureml.core import Workspace
ws = Workspace.from_config()
```

Now create an experiment in your workspace. An experiment is another foundational

cloud resource that represents a collection of trials (individual model runs). In this tutorial you use the experiment to create runs and track your model training in the Azure Machine Learning studio. Parameters include your workspace reference, and a string name for the experiment.

In [ ]:
```
from azureml.core import Experiment
experiment = Experiment(workspace=ws, name="diabetes-experiment")
```

Load data and prepare for training

For this tutorial, you use the diabetes data set, which uses features like age, gender, and BMI to predict diabetes disease progression. Load the data from the Azure Open Datasets class, and split it into training and test sets using `train_test_split()` . This function segregates the data so the model has unseen data to use for testing following training.

In [ ]:
```
from azureml.opendatasets import Diabetes
from sklearn.model_selection import train_test_split

x_df = Diabetes.get_tabular_dataset().to_pandas_dataframe().dropna()
y_df = x_df.pop("Y")

X_train, X_test, y_train, y_test = train_test_split(x_df, y_df, test_size=0.2,
```

Train a model

Training a simple scikit-learn model can easily be done locally for small-scale training, but when training many iterations with dozens of different feature permutations and hyperparameter settings, it is easy to lose track of what models you've trained and how you trained them. The following design pattern shows how to leverage the SDK to easily keep track of your training in the cloud.

Build a script that trains ridge models in a loop through different hyperparameter alpha values.

In [ ]:
```
from sklearn.linear_model import Ridge
from sklearn.metrics import mean_squared_error
import joblib
import math

alphas = [0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1.0]

for alpha in alphas:
    run = experiment.start_logging()
    run.log("alpha_value", alpha)

    model = Ridge(alpha=alpha)
    model.fit(X=X_train, y=y_train)
    y_pred = model.predict(X=X_test)
    rmse = math.sqrt(mean_squared_error(y_true=y_test, y_pred=y_pred))
```

```
    run.log("rmse", rmse)

    model_name = "model_alpha_" + str(alpha) + ".pkl"
    filename = "outputs/" + model_name

    joblib.dump(value=model, filename=filename)
    run.upload_file(name=model_name, path_or_stream=filename)
    run.complete()
```

The above code accomplishes the following:

1. For each alpha hyperparameter value in the `alphas` array, a new run is created within the experiment. The alpha value is logged to differentiate between each run.
2. In each run, a Ridge model is instantiated, trained, and used to run predictions. The root-mean-squared-error is calculated for the actual versus predicted values, and then logged to the run. At this point the run has metadata attached for both the alpha value and the rmse accuracy.
3. Next, the model for each run is serialized and uploaded to the run. This allows you to download the model file from the run in the studio.
4. At the end of each iteration the run is completed by calling `run.complete()` .

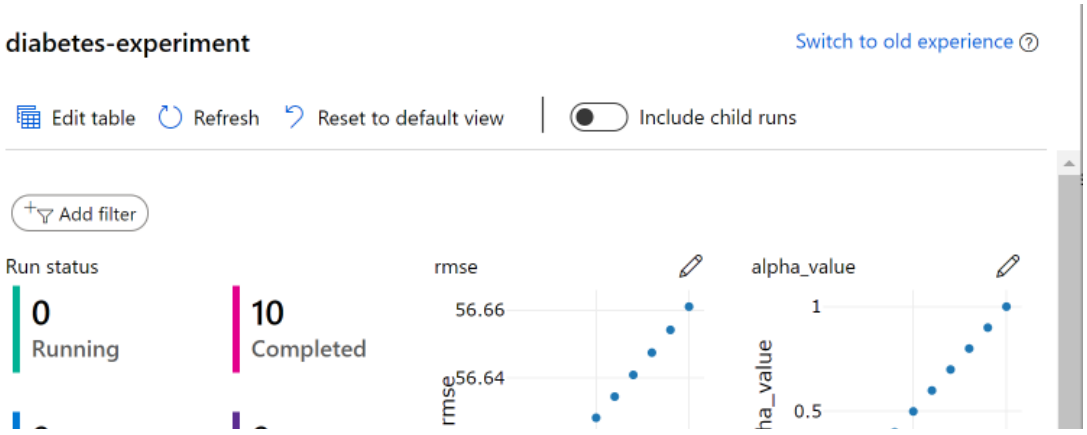
After the training has completed, call the `experiment` variable to fetch a link to the experiment in the studio.

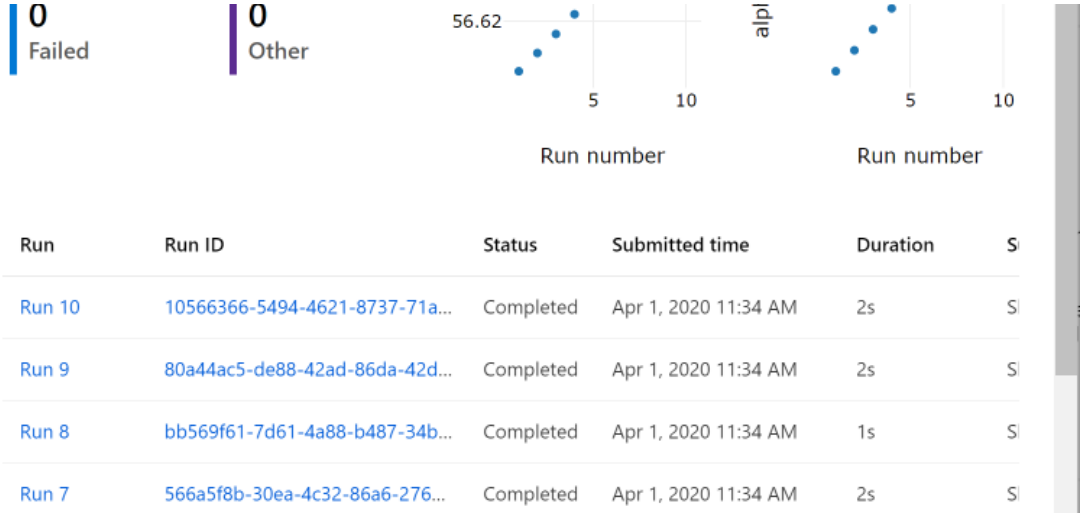
In [ ]:
```
experiment
```

View training results in studio

Following the **Link to Azure Machine Learning studio** takes you to the main experiment page. Here you see all the individual runs in the experiment. Any custom-logged values (`alpha_value` and `rmse` , in this case) become fields for each run, and also become available for the charts and tiles at the top of the experiment page. To add a logged metric to a chart or tile, hover over it, click the edit button, and find your custom-logged metric.

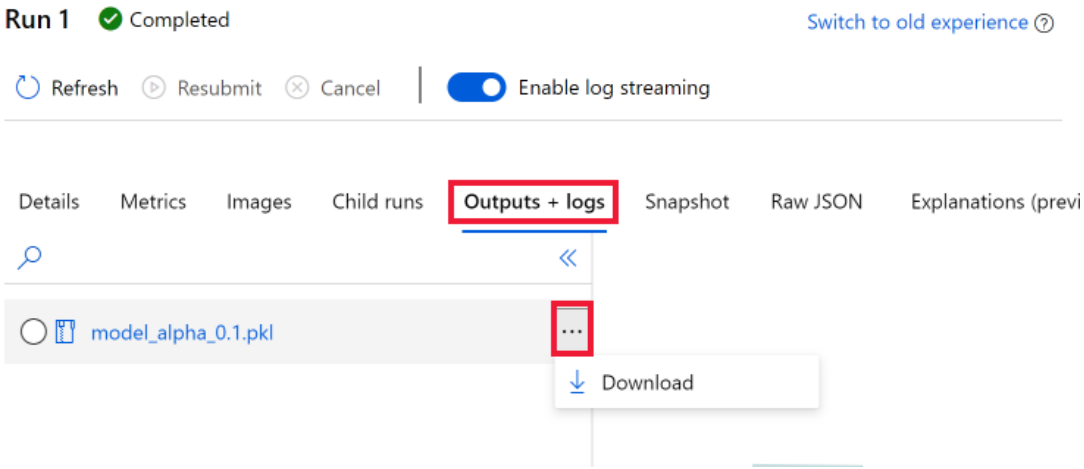
When training models at scale over hundreds and thousands of runs, this page makes it easy to see every model you trained, specifically how they were trained, and how your unique metrics have changed over time.





Run	Run ID	Status	Submitted time	Duration	S
Run 10	10566366-5494-4621-8737-71a...	Completed	Apr 1, 2020 11:34 AM	2s	S
Run 9	80a44ac5-de88-42ad-86da-42d...	Completed	Apr 1, 2020 11:34 AM	2s	S
Run 8	bb569f61-7d61-4a88-b487-34b...	Completed	Apr 1, 2020 11:34 AM	1s	S
Run 7	566a5f8b-30ea-4c32-86a6-276...	Completed	Apr 1, 2020 11:34 AM	2s	S

Select a run number link in the `RUN NUMBER` column to see the page for an individual run. The default tab **Details** shows you more-detailed information on each run. Navigate to the **Outputs + logs** tab, and you see the `.pkl` file for the model that was uploaded to the run during each training iteration. Here you can download the model file, rather than having to retrain it manually.



Get the best model

In addition to being able to download model files from the experiment in the studio, you can also download them programmatically. The following code iterates through each run in the experiment, and accesses both the logged run metrics and the run details (which contains the run_id). This keeps track of the best run, in this case the run with the lowest root-mean-squared-error.

In [ ]:

```
minimum_rmse_runid = None
minimum_rmse = None

for run in experiment.get_runs():
    run_metrics = run.get_metrics()
    run_details = run.get_details()
    # each logged metric becomes a key in this returned dict
    run_rmse = run_metrics["rmse"]
    run_id = run_details["runId"]
```

```
    if minimum_rmse is None:
        minimum_rmse = run_rmse
        minimum_rmse_runid = run_id
    else:
        if run_rmse < minimum_rmse:
            minimum_rmse = run_rmse
            minimum_rmse_runid = run_id

print("Best run_id: " + minimum_rmse_runid)
print("Best run_id rmse: " + str(minimum_rmse))
```

Use the best run id to fetch the individual run using the `Run` constructor along with the experiment object. Then call `get_file_names()` to see all the files available for download from this run. In this case, you only uploaded one file for each run during training.

In [ ]:
```
from azureml.core import Run
best_run = Run(experiment=experiment, run_id=minimum_rmse_runid)
print(best_run.get_file_names())
```

Call `download()` on the run object, specifying the model file name to download. By default this function downloads to the current directory.

In [ ]:
```
best_run.download_file(name="model_alpha_0.1.pkl")
```

Clean up resources

Do not complete this section if you plan on running other Azure Machine Learning service tutorials.

Stop the notebook VM

If you used a cloud notebook server, stop the VM when you are not using it to reduce cost.

1. In your workspace, select **Compute**.
2. Select the **Notebook VMs** tab in the compute page.
3. From the list, select the VM.
4. Select **Stop**.
5. When you're ready to use the server again, select **Start**.