# Track model training in notebooks with MLflow

**You can use MLflow in a notebook to track any models you train. As you'll run this notebook with an Azure Machine Learning compute instance, you don't need to set up MLflow: it's already installed and integrated.**

**You'll prepare some data and train a model to predict diabetes. You'll use autologging, and custom logging to explore how you can use MLflow in notebooks.**

## 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)
```

## Configure MLflow

**As you're running this notebook on a compute instance in the Azure Machine Learning studio, you don't need to configure MLflow.**

**Still, it's good to verify that the necessary library is indeed installed.**

> **Note: If the mlflow library is not installed, run `pip install mlflow` to install it.**

In [ ]:

```
pip show mlflow
```

## Prepare the data

**You'll train a diabetes classification model. The training data is stored in the data folder as diabetes.csv.**

**First, let's read the data:**

In [ ]:

```
import pandas as pd

print("Reading data...")
df = pd.read_csv('./data/diabetes.csv')
df.head()
```

**Next, you'll split the data into features and the label (Diabetes):**

In [ ]:

```
print("Splitting data...")
X, y = df[['Pregnancies','PlasmaGlucose','DiastolicBloodPressure','TricepsThickness','Ser
umInsulin','BMI','DiabetesPedigree','Age']].values, df['Diabetic'].values
```

In [ ]:

```
from sklearn.model_selection import train_test_split

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

**You now have four dataframes:**

- `X_train`: **The training dataset containing the features.**
- `X_test`: **The test dataset containing the features.**
- `y_train`: **The label for the training dataset.**
- `y_test`: **The label for the test dataset.**

**You'll use these to train and evaluate the models you'll train.**

## Create an MLflow experiment

**Now that you're ready to train machine learning models, you'll first create an MLflow experiment. By creating the experiment, you can group all runs within one experiment and make it easier to find the runs in the studio.**

In [ ]:

```
import mlflow
experiment_name = "mlflow-experiment-diabetes"
mlflow.set_experiment(experiment_name)
```

## Train and track models

**To track a model you train, you can use MLflow and enable autologging. The following cell will train a classification model using logistic regression. You'll notice that you don't need to calculate any evaluation metrics because they're automatically created and logged by MLflow.**

In [ ]:

```
from sklearn.linear_model import LogisticRegression

with mlflow.start_run():
    mlflow.sklearn.autolog()
```

```
    model = LogisticRegression(C=1/0.1, solver="liblinear").fit(X_train, y_train)
```

**You can also use custom logging with MLflow. You can add custom logging to autologging, or you can use only custom logging.**

**Let's train two more models with scikit-learn. Since you ran the `mlflow.sklearn.autolog()` command before, MLflow will now automatically log any model trained with scikit-learn. To disable the autologging, run the following cell:**

In [ ]:

```
mlflow.sklearn.autolog(disable=True)
```

**Now, you can train and track models using only custom logging.**

**When you run the following cell, you'll only log one parameter and one metric.**

In [ ]:

```
from sklearn.linear_model import LogisticRegression
import numpy as np

with mlflow.start_run():
    model = LogisticRegression(C=1/0.1, solver="liblinear").fit(X_train, y_train)

    y_hat = model.predict(X_test)
    acc = np.average(y_hat == y_test)

    mlflow.log_param("regularization_rate", 0.1)
    mlflow.log_metric("Accuracy", acc)
```

**The reason why you'd want to track models, could be to compare the results of models you train with different hyperparameter values.**

**For example, you just trained a logistic regression model with a regularization rate of 0.1. Now, train another model, but this time with a regularization rate of 0.01. Since you're also tracking the accuracy, you can compare and decide which rate results in a better performing model.**

In [ ]:

```
from sklearn.linear_model import LogisticRegression
import numpy as np

with mlflow.start_run():
    model = LogisticRegression(C=1/0.01, solver="liblinear").fit(X_train, y_train)

    y_hat = model.predict(X_test)
    acc = np.average(y_hat == y_test)

    mlflow.log_param("regularization_rate", 0.01)
    mlflow.log_metric("Accuracy", acc)
```

**Another reason to track your model's results is when you're testing another estimator. All models you've trained so far used the logistic regression estimator.**

**Run the following cell to train a model with the decision tree classifier estimator and review whether the accuracy is higher compared to the other runs.**

In [ ]:

```
from sklearn.tree import DecisionTreeClassifier
import numpy as np

with mlflow.start_run():
    model = DecisionTreeClassifier().fit(X_train, y_train)
```

```
    y_hat = model.predict(X_test)
    acc = np.average(y_hat == y_test)

    mlflow.log_param("estimator", "DecisionTreeClassifier")
    mlflow.log_metric("Accuracy", acc)
```

**Finally, let's try to log an artifact. An artifact can be any file. For example, you can plot the ROC curve and store the plot as an image. The image can be logged as an artifact.**

**Run the following cell to log a parameter, metric, and an artifact.**

An artifact is a byproduct of software development that helps describe the architecture, design and function of software.

In [ ]:

```
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import roc_curve
import matplotlib.pyplot as plt
import numpy as np

with mlflow.start_run():
    model = DecisionTreeClassifier().fit(X_train, y_train)

    y_hat = model.predict(X_test)
    acc = np.average(y_hat == y_test)

    # plot ROC curve
    y_scores = model.predict_proba(X_test)

    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_param("estimator", "DecisionTreeClassifier")
    mlflow.log_metric("Accuracy", acc)
    mlflow.log_artifact("ROC-Curve.png")
```

**Review the model's results on the Jobs page of the Azure Machine Learning studio.**

- **You'll find the parameters under Params in the Overview tab.**
- **You'll find the metrics under Metrics in the Overview tab, and in the Metrics tab.**
- **You'll find the artifacts in the Outputs + logs tab.**