PHASE 4

**CONTINUE BUILDING THE PROJECT BY DEPLOYING THE MODEL AND INTEGRATING IT INTO APPLICATION**

To solve a problem by training the machine learning models, TrueFoundry helps you track different experiments and makes it easy and intuitive to deploy models with best practices and make it available for public use in a matter of minutes.

In this example, we train a model that can classify a flower of the iris genus into one of three species based on size measurements of its petal and sepal.

The iris dataset contains three different species :

* Iris Setosa
* Iris Versicolor
* Iris Virginica

TrueFoundry provides two libraries for simplifying your ML workflows:

**MLFoundry**

We shall use 5 different APIs from MLFoundry in this example. They are:

1. **log\_params** - use it to log hyper-parameters of the current experiment
2. **log\_dataset** - used to log the entire dataset
3. **log\_metrics** - log metrics like accuracy scores, f1 scores
4. **set\_tags** - add tags to your experiment for easy filtering later on
5. **log\_model** - to save a model including the trained weights

### ServiceFoundry

Open an IPython notebook - you can either use Jupyter running locally on your machine or a Google Colab notebook that runs on the cloud.

Install required libraries.

!pip install mlfoundry

!pip install pandas

!pip install sklearn

Login to TrueFoundry. Create and copy an API key from the settings page. Use this API key to initialise the MLFoundry client and create a run. A run is an entity that represents a single experiment.

import mlfoundry as mlf

client = mlf.get\_client(api\_key='<TFY\_API\_KEY>')

run = client.create\_run('iris-classifier')

Fetch the Iris dataset using the sklearn.dataset module. We then divide it into test and train datasets.

module. We then divide it into test and train datasets.

from sklearn import datasets

from sklearn.model\_selection import train\_test\_split

data = datasets.load\_iris()

X = pd.DataFrame(data.data, columns=data.feature\_names)

y = data.target

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, train\_size=0.8, stratify=y, random\_state=42)

Let's take a look at the target names. We will use this to map from the integer output from the model to the actual names of the species

print(data.target\_names)

# Output

# ['setosa' 'versicolor' 'virginica']

Initialise a model. Then use MLFoundry to log the parameters of the model and create some tags for this current experiment run.

from sklearn.svm import SVC

clf = SVC(gamma='scale', kernel='rbf', probability=True, C=1.2)

run.set\_tags({

'framework': 'sklearn',

'task': 'classification'

})

run.log\_params(clf.get\_params())

Next, we train the model on our train dataset. Once the training is complete, we compute the various metrics and log them to MLFoundry using log\_metrices.

clf.fit(X\_train, y\_train)

y\_pred\_train = clf.predict(X\_train)

y\_pred\_test = clf.predict(X\_test)

metrics = {

'train/accuracy\_score': accuracy\_score(y\_train, y\_pred\_train),

'train/f1\_weighted': f1\_score(y\_train, y\_pred\_train, average='weighted'),

'train/f1\_micro': f1\_score(y\_train, y\_pred\_train, average='micro'),

'train/f1\_macro': f1\_score(y\_train, y\_pred\_train, average='macro'),

'test/accuracy\_score': accuracy\_score(y\_test, y\_pred\_test),

'test/f1\_weighted': f1\_score(y\_test, y\_pred\_test, average='weighted'),

'test/f1\_micro': f1\_score(y\_test, y\_pred\_test, average='micro'),

'test/f1\_macro': f1\_score(y\_test, y\_pred\_test, average='macro'),

}

run.log\_metrics(metrics)

If we are happy with the accuracy scores and other metrics, we can choose to deploy the current model. For that, we need to save the model and **copy the current run id**.

run.log\_model(clf, framework=mlf.ModelFramework.SKLEARN)

print(run.run\_id)

run.end()

## Deploying our model as an API service

To deploy the model using ServiceFoundry, we need to create a Python file containing the function that we want to expose as an endpoint.Inside that Python file, we will fetch the model we just trained and saved using the run id, using MLfoundry. Note that API key required by Mlfoundry will be available as the environment variable TFY\_API\_KEY.In your IPython notebook, create a block with the following contents and run it to create a Python file named predict.py We use the Jupyter magic command %%Writefile to create the file in the notebook environment.

%%writefile predict.py

import os

import json

import pandas as pd

import mlfoundry as mlf

client = mlf.get\_client(api\_key=os.environ.get('TFY\_API\_KEY'))

run = client.get\_run('79e71482643f46dfa5bfef256dba5dc5') # replace with your run id

model = run.get\_model()

def species(features):

features = json.loads(features)

df = pd.DataFrame.from\_dict([features])

prediction = model.predict(df)[0]

return ['setosa', 'versicolor', 'virginica'][prediction]

Inside the species function, we load the features into a pandad DataFrame and make the prediction using the  model. We translate from the integer class to species names using the target\_names we printed during training.

That's pretty much all the work you'll need to do. Now let's deploy this model as an API service. First, install and import servicefoundry in your notebook. Login to servicefoundry

!pip install servicefoundry

import servicefoundry.core as sfy

sfy.login()

Go to True Foundry Dashboard and create a workspace to deploy the service. Workspace are a way to group together related projects inside TrueFoundry. Once the workspace is created, copy the FQN so we can tell servicefoundry where to deploy the model.

Service foundry library lets you gather all the dependencies of the file you just created using gather\_requirements function.

requirements = sfy.gather\_requirements("predict.py")

Now create a sfy.service object, provide the workspace FQN and deploy it by calling deploy().

auto\_service = sfy.Service("predict.py", requirements, sfy.Parameters(

name="iris-service",

workspace="<workspace-fqn-you-copied>"

))

auto\_service.deploy()

You can track the progress of this deployment on the dashboard. Once the deployment is complete, you can access the deployed service from there and try it out.

The TrueFoundry dashboard also links to metrics and logs that come out-of-the-box with TrueFoundry deployments in the form of Grafana dashboards.