# DP _100 Summary

Follow Microsoft documents too.

Important Links

Beginning

DMOC - https://www.skillpipe.com/#/account/login

Azure Pass Redeem - https://www.microsoftazurepass.com/Home/

Github - https://github.com/MicrosoftLearning/mslearn-dp100

practice

Portal - https://azure.microsoft.com/en-in/features/azure-portal/

Github - https://github.com/MicrosoftLearning/mslearn-dp100

Balance Check -
https://www.microsoftazuresponsorships.com/

For Exam Preparation

Schedule Exam - - https://docs.microsoft.com/en-us/learn/certifications/exams/dp-100

Qubits - https://www.qubits42.com/

GHL - https://theghl.in/Course/8225

Certification - https://query.prod.cms.rt.microsoft.com/cms/api/am/binary/RE2PjDI

Balance - https://www.microsoftazuresponsorships.com/

Feedback link -

https://www.metricsthatmatter.com/url/u.aspx?87295A88E167705586

For queries: -

sourabh.taneja@koenig-solutions.com

Followed:-

—> PPT

—> Instructions (In Github)

—> Python Files (.ipynb in Github)

Model —> Register —> Deployment

# Module 1 (Intro of ML studio and SDK)

## Workspace

**********************************************
**********************************************
***********************

# Azure Machine Learning

Azure Machine Learning is a platform for operating machine learning workloads in the cloud.

Built on the Microsoft Azure cloud platform, Azure Machine Learning enables you to manage:

Scalable Compute                                   Data Storage
ML workflow Orchestration                          Model registration
Metrics and Monitoring                             Model Deployment

#Machine Learning Operations :

- Version Control and tracking
- Continuous Integration and Delivery (CI / CD)
- Continuous Monitoring

# Azure Machine Learning Workspaces:

A workspace is a context for the experiments, data, compute targets, and other assets associated with a machine learning workload.

The assets in a workspace include:

- Compute targets for development, training, and deployment.
- Data for experimentation and model training.
- Notebooks containing shared code and documentation.
- Experiments, including run history with logged metrics and outputs.
- Pipelines that define orchestrated multi-step processes.

- Models that you have trained.

#Workspaces as Azure Resources

MS Azure  Subscription  Resource Group
Workspace

The Azure resources created alongside a workspace include:

- A storage account
- An Application Insights instance
- An Azure Key Vault instance
- A container registry

# Creating a Workspace :

You can create a workspace in any of the following ways:

- In the Microsoft Azure portal, create a new Machine Learning resource, specifying the subscription, resource group and workspace name.

```
    ws = Workspace.create(name='aml-
workspace',

    subscription_id='123456-abc-123...',
                                        resource_group='aml-
resources',
                                         location='eastus'  )

ws = Workspace.get( subscription_id='123456-
abc-123...',
                                        resource_group='aml-
resources',
                                        Workspace = ‘ name’ )
ws = Workspace.from_config()
```

<u>Introduction to the Azure ML SDK</u>  :

Azure Machine Learning (Azure ML) is a cloud-based service for creating and managing machine learning solutions.

<u>Check the Azure ML SDK Version</u>

```
print("Ready to use Azure ML", azureml.core.VERSION)
```

Use the Azure Command Line Interface (CLI) with the Azure Machine Learning CLI extension.

```
az ml workspace create  -g 'aml-resources' -w 'aml-workspace'
```

```
# View Azure ML  Registered Compute Target

for compute_name in ws.compute_targets:
   compute = ws.compute_targets[compute_name]
   print("\t", compute.name, ':', compute.type)
```

# **IDE -**

**Visual Studio -**Azure ML Extension for Visual Studio Code provides a graphical interface for working with assets in an Azure ML workspace.

**Jupyter**

# Role-Based Access Control

- Access to workspace resources is managed through Azure Active Directory (AAD) role-based access control (RBAC).
- A user is
    - authenticated by AAD (Azure Active Directory) and
    - receives an access token,
    - this token is used to request access to individual resources, and
    - access is granted (or denied) based on membership of roles that have permission to perform specific actions.

## LAB 1 :- Create and Explore an Azure Machine Learning Workspace

- **Create an Azure Machine Learning workspace**

Azure portal —> new Machine Learning

- Subscription
- Resource group
- Workspace name
- Region
- Storage account / Key vault / Application insights / Container registry
-

- **Explore Azure Machine Learning studio**

**- Create a compute instance**

Compute Page —> Compute Instance tab

- Region
- Virtual machine type: CPU
- Virtual machine size: Standard_DS11_v2
- Compute name: enter a unique name

**- Clone and run a notebook**

Notebooks page —> Terminal

git clone https://github.com/MicrosoftLearning/mslearn-dp100

PYTHON File

**QUIZ 1 :-**

.

**1.A platform for operating machine learning workloads in the cloud**

**A. Azure Cloud**

**B. Azure Data Storage**

**C. IPL**

**D. Azure Machine Learning**

**2. We can create Resource Group without any Subscription.**

**A. True B. False**

**3. Which of the following Azure resources are created alongside an Azure Machine Learning workspace?**

A. Storage Account. B. Space X Databricks

C. Key Vault. D. Application Insights

**4. It is not cloud-based development workstation in your workspace**

A. Built-in Jupyter,

B. JupyterLab

C. EarthLab

D. RStudio

**5. AAD stands for**

A. Active Azure Directory

B. Azure Active Directory

C. AajTak And DD News

D. Assam Assembly in Deutschland

**6. Azure ML don’t contains this Graphical modelling tools**

**A. Automated ML**

**B. Designer**

**C. Visual Studio Query tool**

**7. Which of them is the right way to connect with workspace?**

**A. Workspace. from_compute( )**

**B.  Workspace. from_config( )**

**C.  Workspace. from_compute_target( )**

**D.  Workspace. Azureml-sdk( )**

**8.** It is a **web-based tool** for managing an Azure ML workspace.

A. Azure Databrick

B. Azure Compute Target

C. Azure Machine Learning studio

D. AutoML

9. A drag and drop interface for "**no code**" machine learning model development, is known as

A. Rishabh Pant

B. AutoML

C. Designer

D. Hyperparameter

10. Azure ML don't enables you to manage

- Model Registration
- Monitoring
- Data Storage

A. True     B. False

# **Module 2 (No-Code Machine Learning)**

## Auto ML & Designer

## Auto ML

*************************************************************************************************************

Automated Machine Learning

**What is Automated Machine Learning?**

**Automated ML in Azure Machine Learning Studio**

1. **Select dataset**
   - Upload files
   - Import from Web
   - Register data source
2. **Configure run**
   - Experiment name
   - Target label
   - Compute
3. **Task type and settings**
   - Classification
   - Regression
   - Time Series

**Configuration and Featurization**

**Configuration Options**

- **Primary metric (used to evaluate the best model)**
- **Explain best model (generates feature importance)**

- **Blocked algorithms (restricts training algorithms)**
- **Exit criterion (enables early-stopping)**
- **Validation (sets cross-validation technique)**
- **Concurrency (sets number of parallel iterations)**

**Lab: Use Automated Machine Learning**

## LAB 2 :- **Use Automated Machine Learning**

**- Configure compute resources - Compute instances / Compute clusters**

**- Create a dataset**

**- Run an automated machine learning experiment**

- **Automated ML** page — Create a new Automated ML run

1. **Select dataset**:

2. **Configure run**:

**3. Task type and settings**

**configuration settings:**

- **Primary metric**: **AUC_Weighted**
- **Blocked algorithms**:
- **Exit criterion**:
    - **Training job time (hours)**: 0.5
    - **Metric score threshold**: 0.90

**Featurization settings:**

- **Enable featurization**

- **Review the best model**

**Details** tab — **Algorithm name** for the best model — **View all other metrics**

**- Deploy a predictive service**

**Details** tab —> **Deploy** button

**Endpoints** page —> **auto-predict-diabetes** —> **Consume** tab ( endpoint / Key)

Data --> AutoML  --> BEST MODEL(metric) --> DEPLOY (ACI) --> ENDPOINT

response = requests.post(  endpoint,  input_json, headers)

## Designer

****************************************************************************************************************************

# Azure ML Designer is a visual interface for creating machine learning pipelines.
# Pipeline consists of Dataset and Modules,i.e., it is a data flow for training a ML model. Pipelines have data preparation,model training, scoring, and evaluation.

# Modules are basically the predefined algorithms that process the dataset.

# Steps for Creating a Pipeline:

1. Select a Data set
2. Data Preparation [Removing Missing Values or Columns,Normalization,Splitting]
3. Model Training
4. Score Model
5. Evaluation
6. Submit
7. Deploy as a Service Endpoint.

LAB 3 :**Use Azure Machine Learning Designer**

- Compute / Dataset
- **Create a designer Training pipeline**

**Designer** page —> Rename —> **Select compute target** —>

**diabetes dataset** module

**Normalize Data** module ==> **ZScore** and edit the columns

**Split Data** module  ==> **Splitting mode** , **Fraction, Random seed**

**Train Model** module ==> **Label column** to **Diabetic**

**Two-Class Logistic Regression** module

**Score Model** module

**Evaluate Model** module

click **Submit**. And new experiment named **mslearn-designer-train-diabetes**, and run it.

**Create an inference pipeline**

**Create inference pipeline** drop-down list, —> click **Real-time inference pipeline**. —> Rename

Modifications :-

- Delete the **diabetes dataset** dataset —> **Enter Data Manually ( i**ncludes feature values without labels)

- Delete **Evaluate Model** module,

-  add an **Execute Python Script** module

Submit the pipeline as a new experiment named **mslearn-designer-predict-diabetes**

click **Deploy**, and deploy a new real-time endpoint,

**Test the web service**

- **Endpoints** page, open the **designer-predict-diabetes** real-time endpoint.
- **Consume** tab, note the **REST endpoint** and **Primary key** values.

## LAB

Data -->Training Pipeline —> Inference Pipeline --> DEPLOY (ACI) --> ENDPOINT

Endpoint Page —> Experiment —> Consume Tab--> ENDPOINT / Key

response = requests.post( endpoint, input_json, headers). Or response = urllib.request.Request(  endpoint, input_json, headers)

**What is Azure Machine Learning Designer?**

**Training Pipelines**

**Inference Pipelines**

**Publishing a Service Endpoint**

**Deploy a Real-Time Pipeline:**

Specify deployment target:

- Azure Container Instance
- Azure Kubernetes Services Inference Compute

Submit new data to an HTTP endpoint for immediate results

**QUIZ :-  2**

**1.** *It*  **enables** you to **try multiple algorithms and preprocessing transformations** with your data.

A. Designer

B. *Automated Machine Learning*

*C. Azure Machine Learning*

*D. Pipeline*

*2.* Automated ML capability supports

A. **supervised ML models - Classification, Regression, Time**

B. Un**supervised ML models**

**C. Both**

**3.** You can **not** use automated ML to train models for

- A. **Classification**
- B. **Regression**
- C. **Time series** forecasting

- D. **Clustering**

**4. You run a pipeline to train a model using the Designer tool. What to do to publish it as a real time inference service?**

**A. Create an inference pipeline from your training pipeline.**

**B. Clone the training pipeline with a different name.**

**C. None**

6. The main benefit of using the designer is that it allows for

**A. efficient development**

**B. more Flexibility than SDK**

**C. “no-code” development**

**D. None**

7. Azure Machine Learning Designer **supports a <u>range of algorithms</u>** for both

- supervised learning (classification and regression),
- unsupervised learning (clustering).

A. True B. False

8. **Scoring** is the process of applying an algorithmic model built from a historical dataset to a new dataset.

A. True B. False

# Module 3 (Running Experiments and Training Models)

## Experiments

**************************************************************************************************************

# Experiment:

- Experiment is a named process , running of a script or a pipeline
- An Experiment is a container of trials that represent multiple model runs.

#Creation of Experiment:

experiment = Experiment (workspace, "My Experiment")

#Run : Individual Executional OR Executable process

Run — individual execution (Run1, Run2….) —> Metric, log file

#Run :- Inline vs Script based

*************************************************

- Inline Based Experiment :

run=exp. start_logging( )

….

run.complete()

- Script base Experiment:

  (ScriptRunConfig())

  run = Run.get_Context()
  [azure.ml.core.Run ]

  …..

run.complete()

-

  Run.get_context() method to retrieve the experiment run context when the script is run.

- exp.submit(script_run_config)

# ScriptRunConfig:

ScriptRunConfig():- associate all required item-Data , Script, Dependencies, ComputeContext(instance, cluster, inference)

DATA —> FOLDER <— Script.py

sc = ScriptRunConfig( folder,  script )

ScriptRunConfig( folder,  script)

ScriptRunConfig( folder,  script,  env, compute_target,.Dataset…)

- run.wait_for_completion()

    Wait for the completion of this run. Returns the status object after the wait.

# Steps of Running the Script File :

*******************************************************************************************************************

1. 1.1.  Create Folder(os.makedirs) And

    1.2.   Copying Data  X

2. Create the Script file

 %%write file  foldername/ scriptfile name

 .....script.....

3.

3.1 Environment —> Register Env

3.2 Get Dataset —> Register Dataset

4. ScriptRunConfig

conf = ScriptRunConfig(folder_name, script.py  , environment, argument (tab_ds))

5. exp.submit( conf )

#Retrieve data from metrics :

Logging Metrics :

run. log:

run.log_list:

run.log_row:

run.log_table:

run.log_image:

# Register model by 2 ways:

- Model.Register(workspace,model-name, model_path,description,tags)
- run.register_model()

#View All registered Model

- Model.list(workspace)

# ML FLow

- It is an open source platform for managing ml process
- It tool that to easily track the progress during the model development and tuning

**# Inline ML Flow**

mlflow.start_run()  : to get the reference of run in ML Flow

mlflow.set_tracking_uri(ws.get_mlflow_tracking_uri( ) )

```
with mlflow.start_run():
    mlflow.log_metric("mymetric",123)
```

**# Script based ML Flow:**

```
    with mlflow.start_run():
        mlflow.log_metric("mymetric",123)
```

**************

ScriptRunConfig(folder, script, env)

**QUIZ 3**

1. **Experiment refers to**

**A.** running of a script or a pipeline

B.passing data in script

C. Adding Script code in Workspace

D. running environment

2. exp.**start_logging()**

**A.** Create log folder  in Workspace.

B. start experiment execution

C. Start an interactive logging session

3. **ScriptRunConfig package**

**A.** Record a named list of values with multiple columns.

B. A package for executing Experiment

C. together the configuration information needed to submit a run

5. **It represent a Jupyter notebook widget used to view the progress of model training.**

A. Experiment Run

B. Workspace

C. RunDetails class

D. Child Runs

6. **How to create an experiment variable ?**

A. Experiment(workspace = ws, name = "my-exp")

B.  ws. Experiment(workspace = ws, name = "my-exp")

C. Experiment.create (workspace = ws, name = "my-exp")

D. Workspace.create_experiment(workspace = ws, name = "my-exp")

7. **What should you do to store the model**

**A.** Save the model as a file in a Compute Instance.

B.Save the experiment script as a notebook.

C. Register the model in the workspace.

D. Create a experiment and run script

8. **What should do to record metrics from each run of the experiment and to retrieve them easily from each run.**

A. Add print statements to the experiment code to print the metrics.

B. Use the log methods of the Run class to rec

C.Save the experiment data in the outputs folder.

D. Create a folder and save data in it

9.**Run object is not used to**

**A.** monitor the execution of a trial

B. log metrics and store output of the trial

C. gather the configuration information

D. analyze results

10. **shutil.copy( ) is used to**

**A.** copy the file in specific folder

B. copy experiment in Workspace

C. Copy only model.pkl from Model Page

# Module 4 (Working with Data)

## Datastore & Dataset

Data —> DataStore. —> Dataset

************************************************************************************************************

Data Stores: - Abstractions for cloud Data sources.

Datastores are attached to workspaces and are used to store connection information to Azure storage services

**Built-in Datastores**

- workspaceblobstore (default)
- workspacefilestore

- azureml_globaldatasets

```
blob_datastore =
Datastore.register_azure_blob_container(workspac
e=ws,

datastore_name=blob_datastore_name,

container_name=container_name,

account_name=account_name,

account_key=account_key)
```

- Manage Data stores:

- List all datastores registered in the current workspace

```
datastores = ws.datastores
for name, datastore in datastores.items():
    print(name, datastore.datastore_type)
```

- Get a specific datastore from current workspace

```
datastore = Datastore. get(ws, datastore_name='your datastore name')
```

- Get Default datastore

```
datastore = ws.get_default_datastore()
```

- Change the default datastore

```
ws. set_default_datastore(new_default_datastore)
```

- Upload files to blob Data store

```
blob_ds.upload_files(src_dir='/files',
                     target_path='/data/files',
                     overwrite=True)
```

DataSets:

versioned packaged data object for experiment (Data drifting i.e.,Data changes from time to time, so we can keep track of every change in data )

Generally based on data store,

Datasets .get_by_name(“ds_name”)

- Types of DataSet:

 - Tabular Data Set: data is read from the dataset as a table(structured).

 - File DataSet: dataset presents a list of file_paths that can read from the file system.(unstructured or process the data at the file level)

- Create  and register DataSet in workspace:

 - For Tabular Data

  tab_ds = Dataset.Tabular.from_delimited_files(path=csv_paths) tab_ds = tab_ds.register(ws,name="csv_table")

- For File Data set

blob_ds = ws . get_default_datastore()

file_ds = Dataset . File. from_files (path=(blob_ds, 'data/files/images/*.jpg'))

file_ds = file_ds. register(workspace=ws, name='img_files')

- **For Tabular data set:**

sc = ScriptRunConfig(source_directory='my_dir',

script='script.py',

arguments=['--ds', tab_ds . as_named_input('my_ds')], #Passing Dataset as parameter in ScriptRunConfig

environment=env)

**In the Script fetching the data**

parser.add_argument('--ds', type=str, dest='ds_id')

```
args = parser.parse_args()
run = Run.get_context()
dataset = run.input_datasets['my_ds']
data = dataset.to_pandas_dataframe()
```

- **For File dataset:**

```
sc = ScriptRunConfig(source_directory='my_dir',
                     script='script.py',
                     arguments=['--ds', file_ds .
as_named_input('my_ds').as_download()],
                     environment=env)
```

**In Script fetching the data:**

```
parser.add_argument('--ds', type=str, dest='ds_ref')
args = parser.parse_args()
run = Run.get_context()
dataset = run.input_datasets['my_ds']
```

imgs= glob.glob(dataset + "/*.jpg")

**Create a new Version:**

file_ds.register(workspace=ws, name='img_files', **create_new_version=**True)

Dataset.get_by_name(ws, “dataset”, version = 2)

**QUIZ 4**

1. **This datastore is not created automatically when you create Azure Workspace.**

A. workspaceblobstore

B. azureml_globaldatasets

C. Azure Data Lake stores

D. workspacefilestore

2. **Datastores are**

**A.** abstractions for cloud data sources.

B. abstractions for cloud data set

C. abstractions for cloud data

D. abstractions for cloud MLOps.

**3. How to get a specific DataStore**

A. Datastore.get(ws, dataset_name='blob_data')

B. Datastore.get(ws, subscription_id=‘7cds82d’)

C. Datastore.get(ws, datastore_name='blob_data')

**5. How to get default datastore**

A. ds = ws.set_default_datastore()

B. ds = ws.get_default_datastore()

C. Datastore.get(ws, datastore_name='blob_data')

**6. ______________ are versioned packaged data objects that can be easily consumed in experiments and pipelines.**

A. Datastore

B. workspaceblobstore

C.  Dataset

**7. _________ is recommended way to work with data.**

*A. Datastore*

*B. Dataset*

**8. It is not the type of dataset in Azure Machine Learning**

A. Tabular :

B. CSV

C. File

**9. After registering a dataset, we can not retrieve it by using**

A. datasets dictionary attribute of a Workspace object.

B. get_by_name() method of the Dataset class.

C. get_by_id() method of the Dataset class.

D. get() method of the Dataset class.

10. **You can create a new version of a dataset by registering it with the same name as a previously registered dataset, using**

A. create_new_version attribute

B. create_version attribute

C. create_dataset_version attribute

## Module 5 (Working with Compute)

# Environment & Compute Target

## Environment

*****************************************************************************************

## What are Environments?

- Python code runs in the context of a *virtual environment* that defines
- **version of the Python runtime** to be used
- **installed packages** available to the code.

### Explicitly Creating Environments

### Create from specification file

```
env =
Environment.from_conda_specification(…
conda.yml)
```

## Create from existing conda environment

```
env =
Environment.from_existing_conda_environ
ment(...... 'py36')
```

## Create with specified packages

```
env =
Environment('training_environment')
deps =
CondaDependencies.create(['scikit-
learn','pandas',
'pip'],pip_packages=['azureml-
defaults']

env.python.conda_dependencies = deps
```

## Configuring Environment Containers

```
env.docker.enabled = True
env.docker.base_image='my-base-image'
env.docker.base_image_registry='myregis
try.azurecr.io/myimage'
env.python.interpreter_path =
'/opt/miniconda/bin/python'
```

## Registering Environments

```
env.register(workspace=ws)

Environment.list(workspace=ws)
```

## <u>Reusing Environments</u>

```
training_env =
Environment.get(workspace=ws,
name='training_environment')
script_config =
ScriptRunConfig(source_directory='my_di
r',
script='script.py',

environment=training_env)
```

**Curated Environments** - These include environments that are pre-configured with package dependencies for common frameworks, such **as Scikit-Learn, PyTorch, Tensorflow, and others**.

Azure-ML

## Compute Target

************************************************************************************************************

## 1. Local Compute

- used for most processing tasks.
- **runs** the experiment

## 2. Training Clusters

- Cloud-based cluster managed in workspace
  - use for **training workloads** with **high scalability** requirements
  - **multi-node clusters** of Virtual Machines

## 3 .Attached Compute

- Azure compute resource **outside of a workspace**
- For example:
  - Virtual Machine

### Creating a Compute Cluster

```
compute_config =
AmlCompute.provisioning_configuration(v
m_size='STANDARD_DS11_V2',

max_nodes=4,
vm_priority='lowpriority')
aml_compute = ComputeTarget.create(ws,
compute_name, compute_config)
```

## Attaching Azure Databricks Compute

```
db_config =
DatabricksCompute.attach_configuration(
resource_group=db_resource_group,

workspace_name=db_workspace_name,

access_token=db_access_token)

databricks_compute =
ComputeTarget.attach(ws, compute_name,
db_config)
```

QUIZ 5

**1. In most Python installations, packages are not installed and managed_in environments using**

A.Conda

B. Anaconda

C. pip

**2. For Creating environment from specification file, We should use**

A. Environment.from_existing_conda_environment

B.CondaDependencies.create

C. Environment.from_conda_specification

**3. How can you register an environment**

A. env.register_env(workspace=ws)

B. env.register(workspace=ws)

C. env.register_model(workspace=ws)

**4. To show all environment, use**

A. env.get_all(Workspace = ws)

B. Environment.list(Workspace = ws)

C. Environment.get_all(Workspace = ws)

**5. ______ (Compute target )is use for training workloads with high scalability requirements**

A. Local Compute

B. Attached Compute

C. Training Clusters

# Module 6 (Orchestrating Machine Learning Workflows)

## Pipeline and Publishing Pipeline

## Pipeline

*****************************************************

*****************************************************

**************************

#PIPELINES:

- Contains modules and dataset:

- Workflow of machine learning tasks

        - Each task is a step

    - Steps can be allocated on a specific compute target.

# DIFFERENT PIPELINE STEPS CAN BE IMPLEMENTED:

- PythonScriptStep            - Estimator [ Deprecated ]
              - Data transfer

# Passing data between steps

PipelineData( )

Step 1 = ………

step 2 = ………

training_pipeline = Pipeline(  ws, **[step1,step2]** )

*************************************************************

**LAB**

Folder <— Prep.py / Train.py

Compute Target

Environment

RunConfiguration()

step1 = PythonScriptStep(name='prepare_data', ...) #prep_diabetes.py

step2 = PythonScriptStep(name='train_model', ...) #train_diabetes.py

training_pipeline = Pipeline(  ws, [step1,step2])

pipeline_experiment = Experiment(workspace=ws, name='training-pipeline')

pipeline_run = experiment.submit(training_pipeline )

**Model --> Deployed --> Endpoint (Consume)**

## Publishing Pipeline

****************************************************************************************************************

publish=run . publish_pipeline(name, description, version)

restend=publish.endpoint

# For Consuming the Endpoint,we need these three :

- make http request [**REST ENDPOINT**]
- passing authorize **header**
- json payload specifying the **experiment name**

# For Authorization , **Header**

interactive_auth = InteractiveLoginAuthentication()

auth_header = interactive_auth.get_authentication_header()

# Hitting the **Endpoint**

response = **requests . post**( rest_endpoint, auth_header, json ={"ExperimentName": "run_training_pipeline"}) #Json Payload

run_id = response.json()["Id"]

print(run_id)

# Scheduling Pipelines:- when the service will run again (on specific date or when any data drift occurs)

-**ScheduleRecrrence** and Triggering when DataDrift occur

recur = ScheduleRecurrance(frequency='Day',interval=1)

pipeline_schedule = Schedule.create(ws, name,description,  pipeline id, exp name,  recurr)

**- Event-Driven Workflows**

Azure Ml integrates with Azure Event grid to Support event-driven workflows that go beyond pipelines to create a framework for triggering actions in response to specific events.

# Data Drift Detection:

- Azure Functions                              - Azure Event Hubs

- Azure Logic Apps - Azure Data Factory Pipelines
- Generic Webhooks

regenerate_outputs=True  #Force all Step to re-run

$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$
$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$
$$$$$$$$$$$$$$$$$$

$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$
$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$
$$$$$$$$$$$$$$$$$$

# Module 7 (Deploying and Consuming Models)

# Real time & Batch Inferencing

Model —> Register —> Deployment

## Real time Inferencing

**********************************************
**********************************************
***********************

Real-Time Inferencing :-

Immediate Prediction from new Data

*********************************

1. **Register a trained model (model)**

2. **Define an Inference Configuration (inf)**

- Create a entry/**scoring script** (implement init() and run() functions to load the model and return predictions)
- Create an **environment** (use a Conda configuration file)

**3. Define a Deployment Configuration (aks)**

- **Create a Compute Target (for example: local, Azure Container Instance, AKS cluster)**

**4. Deploy the model as a service**

service = Model.deploy( ws, "name", [model],   inf,  aks)

************************

# Steps for Deploying Real-Time Inferencing Service

**Step 1:- Register Trained Model**

- Model.Register( )
- run.register_model( )

**Step 2:- Define Inference Configuration**

2.1 - Create Scoring Script/Entry Script [init() & run()]

- init() :- for loading the model
- run() :- return the prediction

```
def init():
    global model

    model_path=Model.get_model_path('classification_model')

    model=joblib.load(model_path)

# init will load the model while creating the object of service

def run(raw_data):
    data=np.array(json.loads(raw_data)['data'])
    predictions= model.predict(data)
    return  predictions.tolist()
```

2.2 - Create Environment Config File

```
env=CondaDependencies()
env.add_conda_package('scikit-learn')


env_file=os.path.join(exp_folder,"Environment.yml")

with open(env_file,"w") as f:
    f.write(env.serialize_to_string())
```

inference_configuration=**InferenceConfig**(source_directory, script, environment_file(.yml))

## Step 3:- Define Deployment Configuration

- Create Compute Target (local, Azure Container Instance AKS Cluster)

- For Azure container instance

deploy_configuration=AciWebService.deploy_configuration(cpu_cores=1, memory_gb=1)

- For AKS Web Service

deploy_configuration=AksWebService.deploy_configuration(    cpu_cores=1, memory_gb=1)

deploy_config=LocalWebserice.deploy_configuration()

**Step 4:- Deploy the model as a service**

service = **Model . deploy**( ws,     'my_service', [model],   inference_configuration, deploy_configuration)

# After Deploying Service ,check the service state ( should be healthy)

service.**state**

# For Reviewing the logs of service

service.**get_logs()**

## # Consuming a real-time Inference Service:-

- **Use SDK**

json_data=json.dumps({data: ndarray data})

response=**service.run** (json_data)

predictions=json.loads(response)

- **Use the REST Endpoint :**

- Get the REST Endpoint

endpoint=service.scoring_uri

x_new = [[0.1,2.3,4.1,2.0],[0.2,1.8,3.9,2.1]] # Array of feature vectors json_data = json.dumps({"data": x_new})

request_headers = { 'Content-Type':'application/json' }

response = requests.post( **endpoint, json_data, request_headers )**

predictions = json.loads(response.json())

# LAB :-

**Step 1 :-Train and register a model (model)**

**Deploy the model as a web service**

**Step 2 :- Inference Config**

Step 2.1:Scoring Script file

Step 2.2: Environment

InferenceConfig( script_file, env_file)

**Step 3 :-Deployment Configuration**

deployment_config = AciWebservice.deploy_configuration(cpu_cores = 1, memory_gb = 1)

**Step 4 :-Deploy Model**

**service = Model.deploy(ws, service_name, [model], inference_config, deployment_config)**

**Use the web service**

SDK ==> service.run(input_data = input_json)

ENDPOINT ==> request.post( ENDPOINT, HEADER, INPUT)

## Batch Inferencing

**********************************************
**********************************************
***********************

-Asynchronous prediction from Batched data implemented as a pipeline

#Steps for Batch Inferencing Pipeline:-

**Step 1:- Register the Model**

- Model.register()
- run.register_model()

**Step 2:- Create Scoring Script**

- init()

  Is used to load the model.

```
def init():
    global model
model_path=Model.get_model_path("classification_mod
el" )
    model=joblib.model(model_path)
```

- run(mini_batch)

  To generate predictions from each mini_batch and return results.

```
    def run(mini_batch):
        # This runs for each batch
resultList = []
```

```
 # process each file in the batch
 for f in mini_batch:
        # Read the comma-delimited data into an array
        data = np.genfromtxt(f, delimiter=',')
        # Reshape into a 2-dimensional array for prediction
(model expects multiple items)
        prediction = model.predict(data.reshape(1, -1))
        # Append prediction to results
        resultList.append("{}:
{}".format(os.path.basename(f), prediction[0]))
          return resultList
```

**ENVIRONMENT**

**Step 3:- Create pipeline with a ParellelRunStep to run the Script**

- Define File dataset input for the batch data

```
batch_dataset=ws.datasets('batch-data')
```

- Define a pipelineData reference for the output folder

```
output_dir=PiplineData(name,
datastore[batch_dataset], output_path)
```

- Configure with an output action of append row so all results are collated in parellel_run.txt

```
output_action="append_row"
```

```
parallel_run_config = ParallelRunConfig(
source_directory='batch_scripts',
entry_script    ="batch_scoring_script.py",
environment   =batch_env,
compute_target    =aml_cluster,

mini_batch_size   ="5",
error_threshold    =10,
output_action  ="append_row",
node_count     =4)
```

```
parallelrun_step = ParallelRunStep(
    name='batch-score-diabetes',
    parallel_run_config=parallel_run_config,

inputs=[batch_data_set.as_named_input('diabetes_batch')
],
    output=output_dir,
    arguments=[],
    allow_reuse=True
)
```

**Step 4:- Run the Pipeline and Retrieve the step Output**

```
  pipeline=Pipeline(ws,step=[paralell_step])
  pipeline_run=Experiment(ws,name).submit(pipeline)
```

## # Publish the Pipeline  and get the endpoint:

```
published_pipeline = pipeline_run. publish_pipeline
(name,description,version)
rest_endpoint=published_pipeline.endpoint

response = requests.post(rest_endpoint,
                         headers=auth_header,
                         json={"ExperimentName": "mslearn-
diabetes-batch"})
run_id = response.json()["Id"]
PipelineRun(Exp, run_id)


# Get the outputs from the first (and only) step
prediction_run = next(pipeline_run.get_children())
prediction_output =
prediction_run.get_output_data('inferences')

for root, dirs, files in os.walk('results'):
    for file in files:
        if file . endswith('parallel_run_step.txt'):
            result_file = os.path.join(root,file)
```

# # CI/CD [Continuous Integration and Continuous Delivery ]

CI/CD is a method to frequently deliver apps to customers by introducing automation into the stages of app development. The main concepts attributed to CI/CD are continuous integration, continuous delivery, and continuous deployment. CI/CD is a solution to the problems integrating new code can cause for development and operations teams.

Continuous deployment (the other possible "CD") can refer to automatically releasing a developer's changes from the repository to production, where it is usable by customers.

<u>GitHub Actions</u>: When using GitHub as a source control repository, you can use GitHub actions to automate processes.

e.g. Use the Azure Machine Learning Compute action to connect to a compute target in Azure Machine Learning, If the compute target exists, the action will connect to it. Otherwise, the action will create a new compute target.

Azure ML action

- aml-workspace - Connects to or creates a new workspace
- aml-compute - Connects to or creates a new compute target in Azure Machine Learning
- aml-run - Submits a ScriptRun, or a Pipeline to Azure Machine Learning
- aml-registermodel - Registers a model to Azure Machine Learning
- aml-deploy - Deploys a model and creates an endpoint for the model

$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$
$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$
$$$$$$$$$$$$$$$$$$

$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$
$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$
$$$$$$$$$$$$$$$$$$

# Module 8 (Training Optimal Models)

## Hyperparameter & AutoML

## Hyperparameter

**************************************************
**************************************************
***********************

- Hyper parameter values makes important to select the best possible values for your particular data and predictive performance goals.
- Hyperparameter tuning is accomplished by training the multiple models, using the same algorithm and training data but different hyperparameter values.

## Search Space

- Discrete Hyperparameters
  - Choice, qnormal, quniform, qlognormal, qloguniform
- Continuous Hyperparameters
  - normal, uniform, lognormal, loguniform

```
param_space = { '--batch_size': choice(30, 50, 100),
              '--learning_rate': normal(10, 3) }
```

  -

## Hyperparameter Sampling

- Grid Sampling
  - Tries every combination of discrete hyperparameter values

- Can only be used when all hyperparameters are discrete

```
param_space = {
              '--batch_size‘    : choice(16, 32, 64),
              '--learning_rate': choice(1349342, 2, 3) }
param_sampling = GridParameterSampling(param_space)

reg_rate - 0.1
```

- Random Sampling

  - Randomly selects hyperparameter values

  - Can be used with discrete and continuous hyperparameter combinations

```
    param_space = {
        '--batch_size': choice(16, 32, 64),
        '--learning_rate': normal(10, 3) }
        param_sampling =
RandomParameterSampling(param_space)
```

- Bayesian Sampling

- Selects hyperparameter values based on performance of previous selection
- Can only be used with **choice, uniform, and quniform** hyperparameters

```
param_space = {
    '--batch_size': choice(16, 32, 64),
    '--learning_rate': uniform(0.5, 0.1) }

param_sampling =
BayesianParameterSampling(param_space)
```

**Early Termination Policy**

Evaluate primary metric at intervals and compare to previous runs

- Bandit Policy:
    - Stop if metric underperforms the best run so far by a specified margin
- Median Stopping:
    - Stop if metric is worse than median of running averages

- Truncation Selection:

  Stop if metric is in the worst X% of all runs at the same interval

Tuning Hyperparameters with Hyperdrive

```
hyperdrive =
HyperDriveConfig(run_config=script_config,

hyperparameter_sampling=param_sampling,

                    policy=stop_policy,
                    primary_metric_name='Accuracy',

primary_metric_goal=PrimaryMetricGoal.MAXIMIZE,
                    max_total_runs=6,
                    max_concurrent_runs=4)

hyperdrive_run = experiment.submit(config=hyperdrive)
```

```
for child_run in hyperdrive_run.get_children():
    print(child_run.id, child_run.get_metrics())
```

- **List all runs by performance metric:**

for child_run in hyperdrive_run.**get_children_sorted_by_primary_metric():**

```
    print(child_run)
```

- **Find the best-performing run**:

best_run = hyperdrive_run.**get_best_run_by_primary_metric()**

*************************************************************************************************************************

# LAB

**Prepare Data**

**Prepare a training script**

**- Folder**

**- Create Script file in folder**

**- Create Compute**

**- Environment**

**- Dataset**

**- script_config = ScriptRunConfig ( folder, script, dataset, compute, env)**

**- ss = Sampling ( Search Space)**

**- HyperDriveConfig( script_config, ss)**

**Determine the best performing run**

## AutoML

**********************************************
**********************************************
***********************

- Automated Machine Learning enables you to try multiple algorithms and preprocessing transformations with your data.
- Default set of algorithms for machine learning tasks:
    - Classification
    - Regression
    - Time Series Forecasting

```
automl_config = AutoMLConfig(name='Automated ML Experiment',
                    task='classification',
                    primary_metric = 'AUC_weighted',
                    compute_target=aml_compute,
                    training_data = train_dataset,
                    validation_data = test_dataset,
                    label_column_name='Label',
                    featurization='auto',
                    iterations=12,
                    max_concurrent_iterations=4)

automl_run = automl_experiment.submit(automl_config)
```

**Find the best-performing run**

```
      best_run, fitted_model = automl_run.get_output()
      best_run_metrics = best_run.get_metrics()
      for metric_name in best_run_metrics:
         metric = best_run_metrics[metric_name]
         print(metric_name, metric)
```

**View model pipeline details**

```
for step_ in fitted_model.named_steps:
   print(step)
```

*********************************************
*********************************************
**********************

AUTOML LAB

- **Prepare dataset**

**- Split data**

- **Prepare compute**

- **AutoMLConfig( … )**

**- Run an automated machine learning experiment**

**-View child run details**

$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$
$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$
$$$$$$$$$$$$$$$$$$

# Module 9 (Responsible Machine Learning)

## Differential Privacy, Model Interpratibility & Fairness

*****************************************************************************************************************

- Differential Privacy

data includes sensitive personal details that should be kept private.so analysis adds random "noise" to the data.

Epsilon -the privacy Loss Parameter

#Perform Analysis

  Class :- SmartNoise

data

- Upper bounds and Lower Bounds (eg- age ( 0, 120 )
- Sample Space (no of rows)
- epsilon :- non negative data , generally values between 0 and 1

- sn.dp_mean() / Actual_mean
- sn.dp_histogram() / / Actual_histo
-sn.dp_covariance() / / Actual_cov

Sql interpretation:-
- privaterearder : for holding metadata
- pandasreader : for holding actual data

privaterearder . execute( query )

****************************************
****************************************
************************

## Model Interpratibility

**************************************************************************************************************************

Module Interpretability:

- to be able to understand how models make predictions
- to be able to  explain the rationale for decisions while identifying and mitigating bias

- Model Interpretability is based on the open source Shapely additive Explanation(SHAP) / LIME

**Global Feature** :- Overall feature importance for all test data

**Local Feature :-** Feature Importance for an individual prediction

tab_explainer = TabularExplainer(model, X_train, features, labels)

explanation = tab_explainer.explain_global(X_train)

client = ExplanationClient.from_run(run)
client.upload_model_explanation(explanation)

explanation = client.download_model_explanation()

# Explainer :

- MimicExplanaier : Global Surrogate model that approximates your model

*explainer = MimicExplainer(model, X_train, LinearExplainableModel,  features=features, classes=labels)*

*explanation = explainer.explain_global(X_test)*

*explain_client = ExplanationClient.from_run(run)*

*explain_client.upload_model_explanation(explanation)*

**Get local feature importance:**

X_explain = X_test[0:2]

```
predictions = model.predict(X_explain)
local_tab_explanation =
tab_explainer.explain_local(X_explain)

local_tab_features =
local_tab_explanation.get_ranked_local_names()
local_tab_importance =
local_tab_explanation.get_ranked_local_values()
**********************************************
**********************************************
***********************
```

Model

ref = TabularExplainer

Explaination = ref.explain_global()

Folder — Script ( Explaination—> Experiment)

Environment

ScriptRunConfig()

# Fairness of Model

****************************************************************************************************************************

**Fairlearn**

1 Dashboards

2 Mitigation Strategies (GridSearch, Exponentiated Gradient, Threshold Optimizer)

3. Integrated with Azure ML

Measure Disparity in Prediction Performance:

When you train a ML Model using a supervised technique, You us metrics achieved against

- Potential causes of disparity

Data imbalance. Some groups may be overrepresented in the training data, or the data may be skewed so that cases within a specific group aren't representative of the overall population.

- Indirect correlation. The sensitive feature itself may not be predictive of the label, but there may be a hidden correlation between the sensitive feature and some other feature that influences the prediction. For example, there's likely a correlation between age and credit history, and there's likely a correlation between credit history and loan defaults. If the credit history feature is not included in the training data, the training algorithm may assign a predictive weight to age without accounting for credit history, which might make a difference to loan repayment probability.

Fairlearn is a Python package that you can use to analyze models and evaluate disparity between predictions and prediction performance for one or more sensitive features.
**********************************************
**********************************************
***********************

Exponentiated Gradient

Grid Search

Threshold Optimizer

Grid Search( Algo ) --> Equalized odds (Constraint)

==> Reduce disparity of Feature Selection

==> Reduce disparity of performance metric

# Module 10 (Monitoring Models)

## App Insight & Data Drift

****************************************************************************************************************

**Application Insights**:

- Application Insights is an application performance management service that enables the capture, storage, and analysis of telemetry data from applications.

**Determine Application Insights**

```
  ws.get_details()['applicationInsights']
```

**Enabling Application Insights:-**

Webservice.**deploy_configuration(**enable_app_insights =True). # (when a new Service is deployed)

or

**service.update (**enable_app_insights=True). #(existing deployed services)

Get Data from Application Insights:

With the help of SDK : -

```
def init():
model=joblib.load(Modl.gt_Modl_path("my_modl"))

def run(raw_data):
    data=json.loads(raw_data)['data']
    perdictions=modl.perdict(data)
    log_txt='Data'+str(data)+ 'Perdictions'
 +str(perdictions)
    print(log_txt)
```

## Data Drift

**********************************************************************************************************************************

Data Drift - change in data profiles between training and inferencing is known as data drift.

### Creating a Data Drift Monitor

monitor = DataDriftDetector.create_from_datasets(ws, 'dataset-drift-detector', **baseline_data_set, target_data_set, ...**)

backfill = monitor.backfill(dt.datetime.now() - dt.timedelta(days=30), dt.datetime.now())

## Data Drift Schedules and Alerts

```
alert_email =
AlertConfiguration('data_scientists@contoso.com')
monitor = DataDriftDetector.create_from_datasets(ws,
'dataset-drift-detector',
                                                 baseline_data_set,
target_data_set,

compute_target=cpu_cluster,
                                                 frequency='Week',
latency=2,
                                                 drift_threshold=.3,

alert_configuration=alert_email)
```

****************************************************************************************************************************

QUIZ 10

1. ______ is an application performance management service that enables the capture, storage, and analysis of telemetry data from applications.

A. Monitoring Model

B. Data Drift

C. Application Insight

D. Querying Log

2. Can you use App Insight to monitor telemetry from many kinds of application, including applications that are not running in Azure.

A. True.   B. False

3.You can't determine the Application Insights resource associated with your workspace by viewing

A. Overview page of the workspace blade in the Azure portal,

B. by using the get_details() of a Workspace object

C. service.update(enable_app_insights=True)

4. If you do not select an existing Application Insights resource, a new one is created in the same resource group as your workspace.

A. True.    B. False

5. We can’t enable Application Insights for a service that is already deployed

A. you can modify the deployment configuration for Azure Kubernetes Service (AKS) based services in the Azure portal

B. you can update any web service.   [ service . update(enable_app_insights=True)]

C. model =joblib.load(Model.get_model_path('my_model'))

6. change in data profiles between training and inferencing is known

A. App Insight

B. Data Drift

C. Real Time Inferencing

7.Azure ML supports data drift monitoring through the use of

A. DataStore

B. Baseline Datastore

C. Datasets.

8.  You can create dataset monitors using

A. visual interface in Azure ML studio,

B. by using the DataDriftDetector class in the Azure ML SDK

C. Using monitor.backfill( )

9. For dataset monitors, you can specify a latency, indicates

A. the number of hours to allow for new data to be collected and added to the target dataset.

B. specify a schedule_start time value to indicate when the data drift run should start

C. specify a schedule on which it should run.

10.You want to capture metrics from a real-time inference service and analyze them using Application Insights. What must you do in the scoring script for the service?

A. Use the Run.log method to log the metrics.

B. Save the metrics in the ./outputs folder.

C. Use a print statement to write the metrics in the STDOUT log.

Regards,

Sourabh Taneja

Mail ID - sourabh.taneja@koenig-solutions.com