Azure Machine Learning

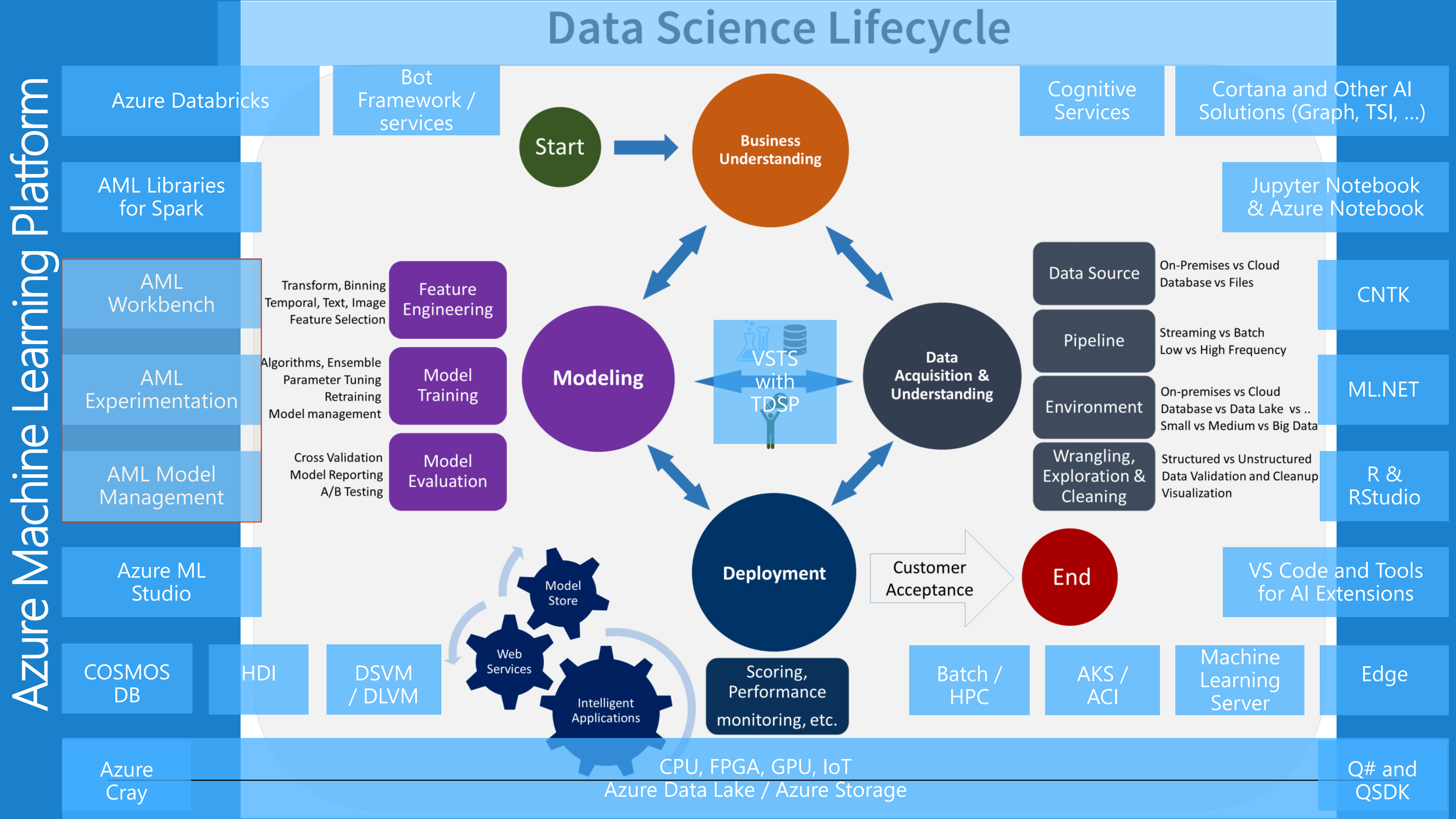
Azure Data Platform

Data Collection	Data Processing	Data Storage	Data Analysis	Presentation
Azure Data Factory	Azure Data Factory	SQL Database	Azure Machine Learning	Power BI
Azure IoT	HDInsight	Table/Blob/File/ Queue Storage	HDInsight	Power BI embedded
Import / Export Service	App Service Cloud Services	Cosmos DB	Azure Data Lake Analytics	SharePoint
SQL Tools	HPC / Batch	SQL DWH	Azure Analysis	App Service Cloud Services
Big Data Tools	Functions	Azure Data Lake Store	DSVM / DLVM	Azure Notebook
Azure Search	Stream Analytics	Blockchain (Bletchley)	Cognitive Services	Excel
Backup/Restore	Azure Data Lake Analytics	Azure DB for MySQL & PostgreSQL	Stream Analytics	QlikView / Tableau
Other Tools (AzCopy)	Azure Database for MySQL / PostgreSQL	VM + SQL Server	Azure Databricks	SQL / VM (SS*S)

The Context

- No Need to have a pre-defined GUI Interface
- End-to-End Lifecycle and processes
- Open to frameworks and tools
- Support Deep Learning frameworks
- Help with Environment isolations
- Better management of models & experiments
- Especially on Tracking and Monitoring
- Deployment to multiple targets
- Help with ease of data preparation
- Automated Machine Learning
- Distributed Training
- Support both for Web Service and Batch modes
- Strong support for Spark (Databricks)
- Support for more training & deployment platforms
- Better Integration with other services

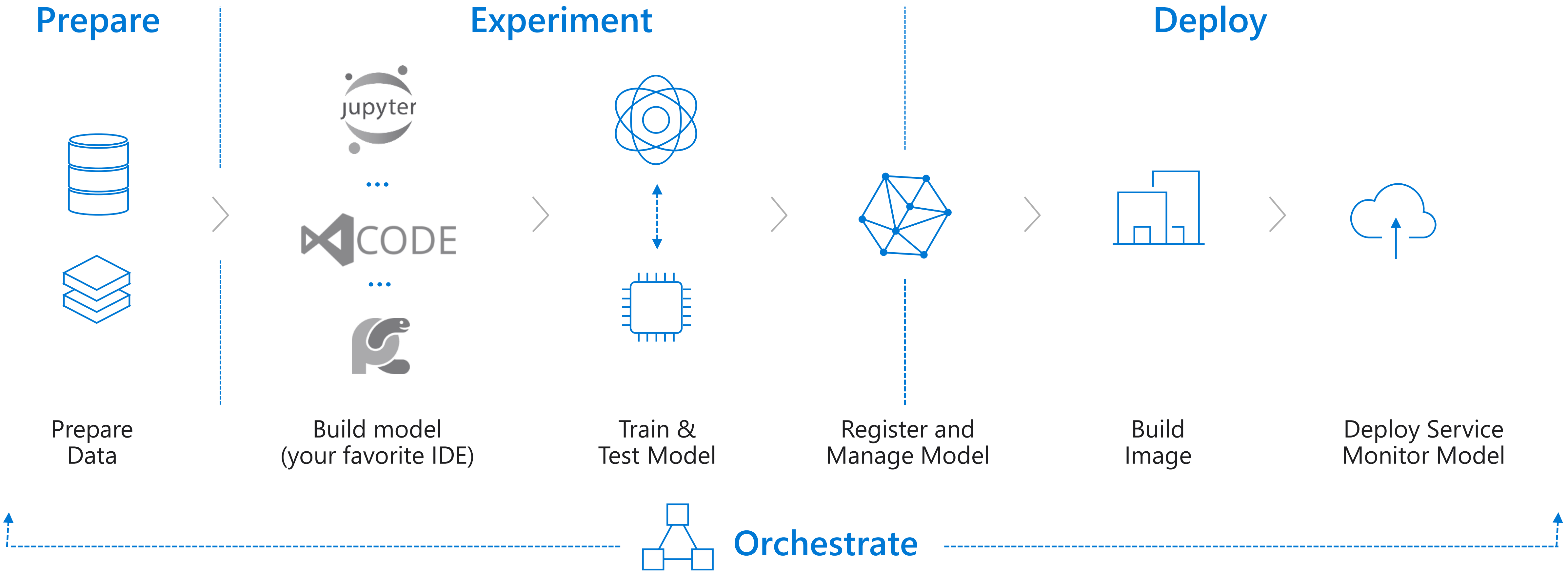
Azure offers a comprehensive AI/ML platform that meets—and exceeds—requirements



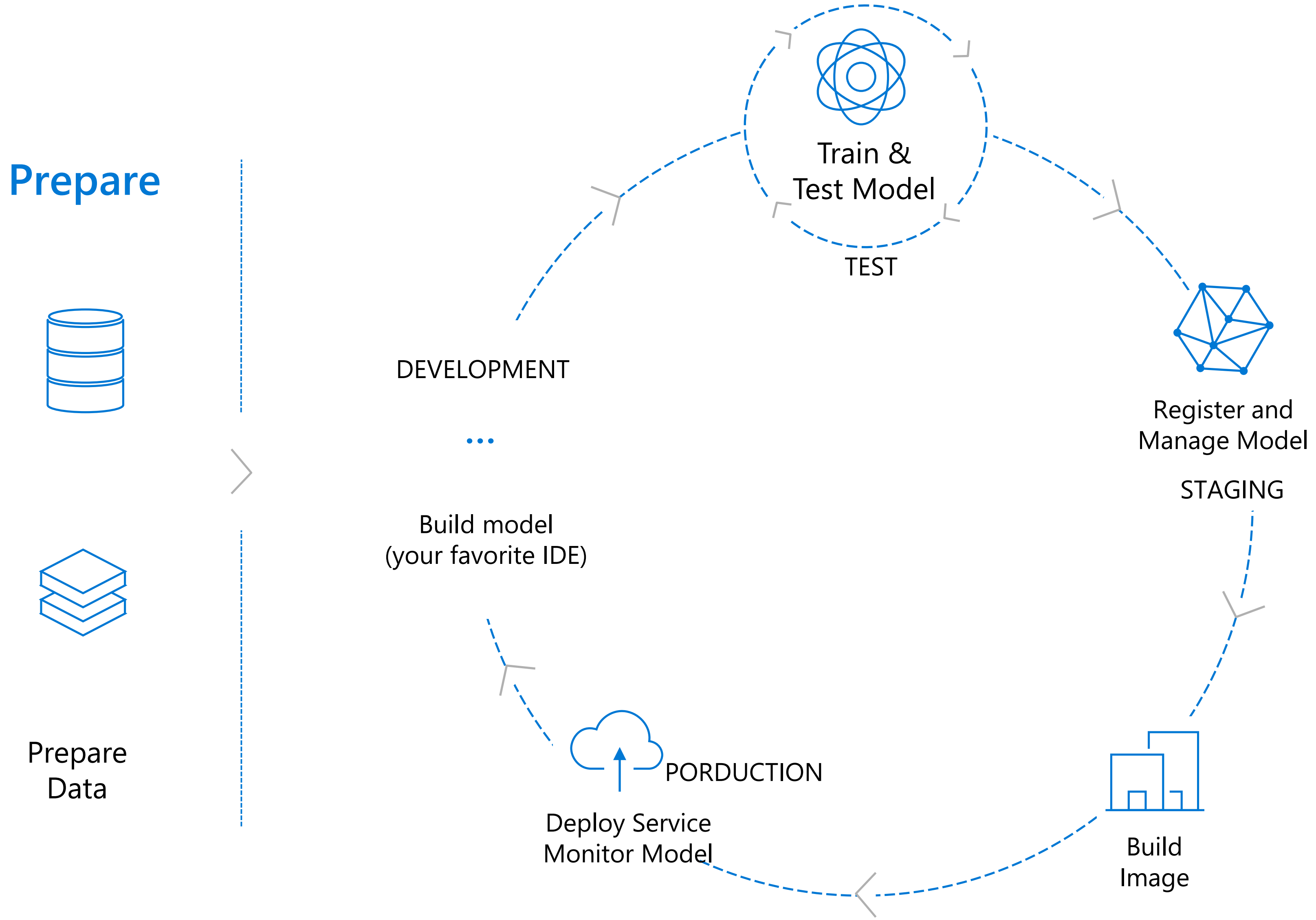
Data Science Lifecycle
Azure Machine Learning Platform
Azure Databricks
Bot Framework / services
Cognitive Services
Cortana and Other AI Solutions (Graph, TSI, ...)
AML Libraries for Spark
Jupyter Notebook & Azure Notebook
AML Workbench
AML Experimentation
AML Model Management
CNTK
ML.NET
R & RStudio
Azure ML Studio
VS Code and Tools for AI Extensions
COSMOS DB
HDI
DSVM / DLVM
Batch / HPC
AKS / ACI
Machine Learning Server
Edge
Azure Cray
Q# and QSDK
CPU, FPGA, GPU, IoT
Azure Data Lake / Azure Storage
Start
Business Understanding
Modeling
VSTS with TDSP
Data Acquisition & Understanding
Deployment
Customer Acceptance
End
Transform, Binning
Temporal, Text, Image
Feature Selection
Feature Engineering
Algorithms, Ensemble
Parameter Tuning
Retraining
Model management
Model Training
Cross Validation
Model Reporting
A/B Testing
Model Evaluation
Data Source
On-Premises vs Cloud
Database vs Files
Pipeline
Streaming vs Batch
Low vs High Frequency
Environment
On-premises vs Cloud
Database vs Data Lake vs ..
Small vs Medium vs Big Data
Wrangling, Exploration & Cleaning
Structured vs Unstructured
Data Validation and Cleanup
Visualization
Model Store
Web Services
Intelligent Applications
Scoring, Performance monitoring, etc.

Machine Learning

Typical E2E Process



DevOps loop for data science



What is Azure Machine Learning service?

Set of Azure Cloud Services

+

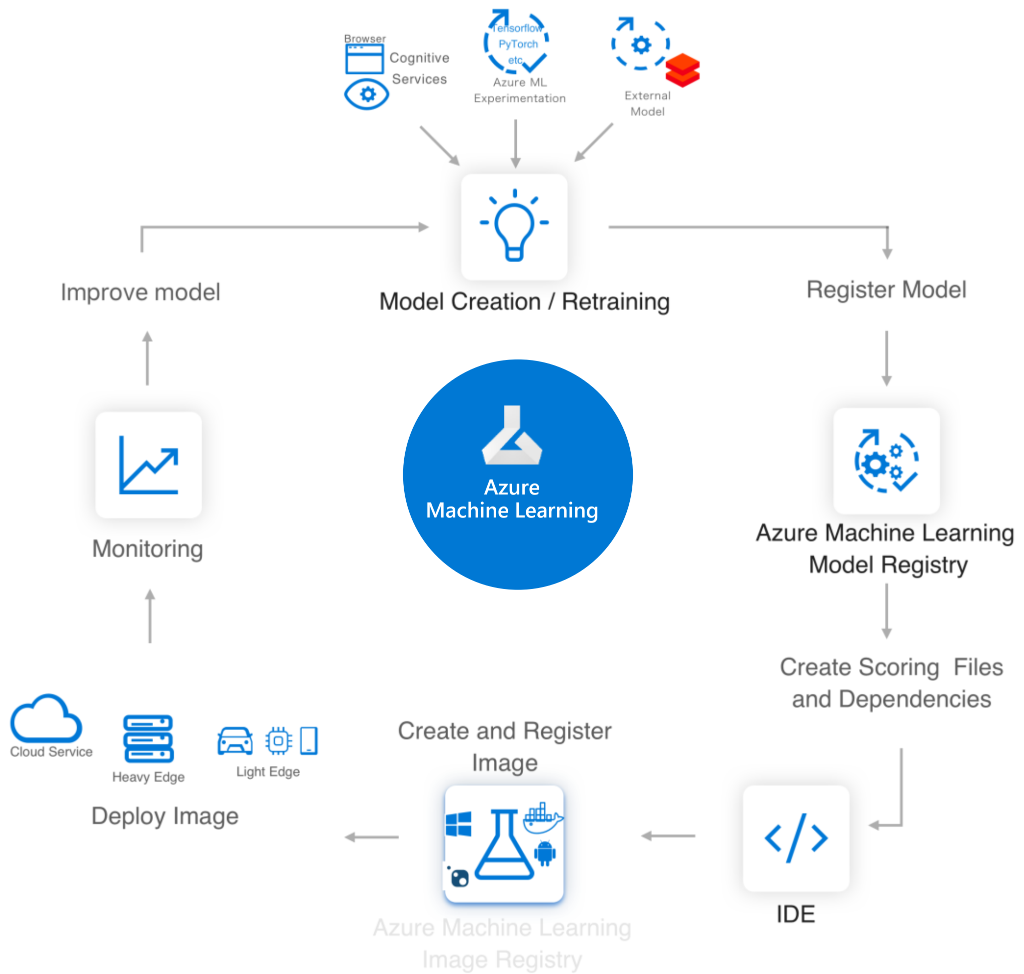
Python SDK

That enables you to:

- ✓ Prepare Data
- ✓ Build Models
- ✓ Train Models
- ✓ Manage Models
- ✓ Track Experiments
- ✓ Deploy Models

Azure ML service

Lets you easily implement this AI/ML Lifecycle



Workflow Steps

Develop machine learning training scripts in Python.

Create and configure a compute target.

Submit the scripts to the configured compute target to run in that environment. During training, the compute target stores run records to a datastore. There the records are saved to an experiment.

Query the experiment for logged metrics from the current and past runs. If the metrics do not indicate a desired outcome, loop back to step 1 and iterate on your scripts.

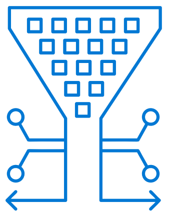
Once a satisfactory run is found, register the persisted model in the model registry.

Develop a scoring script.

Create an Image and register it in the image registry.

Deploy the image as a web service in Azure.

Data Preparation



Multiple Data Sources

SQL and NoSQL databases, file systems, network attached storage and cloud stores (such as Azure Blob Storage) and HDFS.

Multiple Formats

Binary, text, CSV, TS, ARFF, etc. and auto detect file types.

Cleansing

Detect and fix NULL values, outliers, out-of-range values, duplicate rows.

Transformation / Filtering

General data transformation (transforming types) and ML-specific transformations (indexing, encoding, assembling into vectors, normalizing the vectors, binning, normalization and categorization).

Intelligent time-saving transformations

Derive column by example, fuzzy grouping, auto split columns by example, impute missing values.

Custom Python Transforms

Such as new script column, new script filter, transformation partition

Model Building (DEV)

Choice of algorithms

Choice of language

Python

Choice of development tools

Browser-based, REPL-oriented, notebooks such as Jupyter, PyCharm and Spark Notebooks.
Desktop IDEs such as Visual Studio and R-Studio for R development.

Local Testing

To verify correctness before submitting to a more powerful
(and expensive) training infrastructure.

Model Training and Testing

Powerful Compute Environment

Choices include scale-up VMs, auto-scaling scale-out clusters

Preconfigured

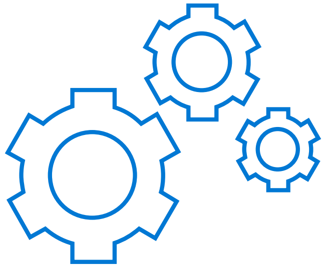
The compute environments are pre-setup with all the correct versions ML frameworks, libraries, executables and container images.

Job Management

Data scientists are able to easily start, stop, monitor and manage Jobs.

Automated Model and Parameter Selection

Solutions are automatically select the best algorithms, and the corresponding best hyperparameters, for the desired outcome.



Model Registration and Management

Containerization

Automatically convert models to Docker containers so that they can be deployed into an execution environment.

Versioning

Assign versions numbers to models, to track changes over time, to identify and retrieve a specific version for deployment, for A/B testing, rolling back changes etc.

Model Repository

For storing and sharing models, to enable integration into CI/CD pipelines.

Track Experiments

For auditing, see changes over time and enable collaboration between team members.

Model Deployment

Choice of Deployment Environments

Single VM, Cluster of VMs, Spark Clusters, Hadoop Clusters, In the cloud, On-premises

Edge Deployment

To enable predictions close to the event source-for quicker response and avoid unnecessary data transfer.

Security

Your data and model is secured. Even when deployed at the edge, the e2e security is maintained.

Monitoring

Monitor the status, performance and security.

Azure Machine Learning: Technical Details

Azure ML service

Key Artifacts

Workspace

- Models
- Experiments / run history
- Pipelines
- Compute Target
- Images
- Deployment
- Data Stores

Azure ML service Artifact

Workspace

The workspace is the **top-level resource** for the Azure Machine Learning service. It provides a centralized place to work with all the artifacts you create when using Azure Machine Learning service.

The workspace keeps a list of compute targets that can be used to train your model. It also keeps a history of the training runs, including logs, metrics, output, and a snapshot of your scripts.

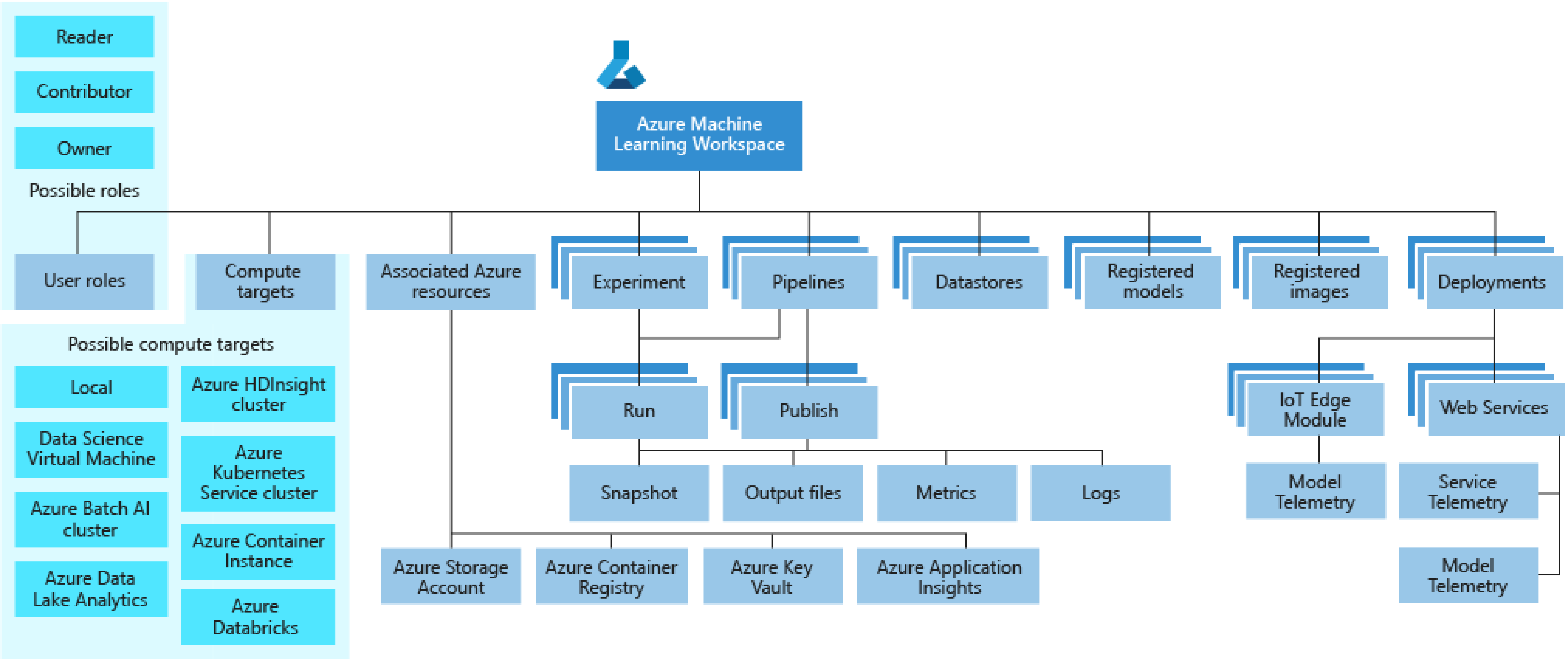
Models are registered with the workspace.

You can create multiple workspaces, and each workspace can be shared by multiple people.

When you create a new workspace, it automatically creates these Azure resources:

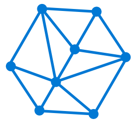
- Azure Container Registry - Registers docker containers that are used during training and when deploying a model.
- Azure Storage - Used as the default datastore for the workspace.
- Azure Application Insights - Stores monitoring information about your model service.
- Azure Key Vault - Stores secrets used by compute targets and other sensitive information needed by the workspace.

Azure ML service Workspace Taxonomy



Azure ML service Artifacts

Models and Model Registry



Model

A machine learning model is an artifact that is created by your training process. You use a model to get predictions on new data.

A model is produced by a **run** in Azure Machine Learning.

> Note: You can also use a model trained outside of Azure Machine Learning.

Azure Machine Learning service is framework agnostic — you can use any popular machine learning framework when creating a model.

A model can be registered under an Azure Machine Learning service workspace

Model Registry

Keeps track of all the models in your Azure Machine Learning service workspace.

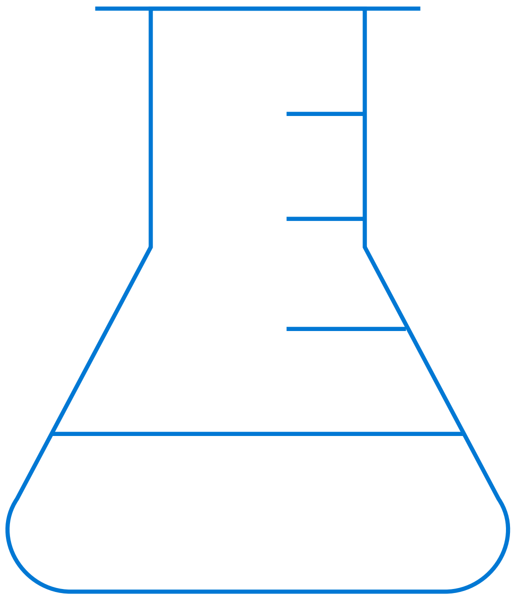
Models are identified by name and version.

You can provide additional metadata tags when you register the model, and then use these tags when searching for models.

You cannot delete models that are being used by an image.

Azure ML Artifacts

Runs and Experiments



Experiment

Grouping of many runs from a given script.

- Always belongs to a workspace.
- Stores information about runs

Run

Produced when you submit a script to train a model. Contains:

- Metadata about the run (timestamp, duration etc.)
- Metrics logged by your script.
- Output files autocollected by the experiment, or explicitly uploaded by you.
- A snapshot of the directory that contains your scripts, prior to the run.

Run configuration

A set of instructions that defines how a script should be run in a given compute target.

Azure ML service Artifacts

Image and Registry

Image contains

1. A model.
2. A scoring script used to pass input to the model and return the output of the model.
3. Dependencies needed by the model or scoring script/application.

Two types of images

1. **FPGA image**: Used when deploying to a field-programmable gate array in the Azure cloud.
2. **Docker image**: Used when deploying to compute targets such as Azure Container Instances and Azure Kubernetes Service.

Image Registry

Keeps track of images created from models.

Metadata tags can be attached to images. Metadata tags are stored by the image registry and can be used in image searches

Azure ML Concept

Model Management

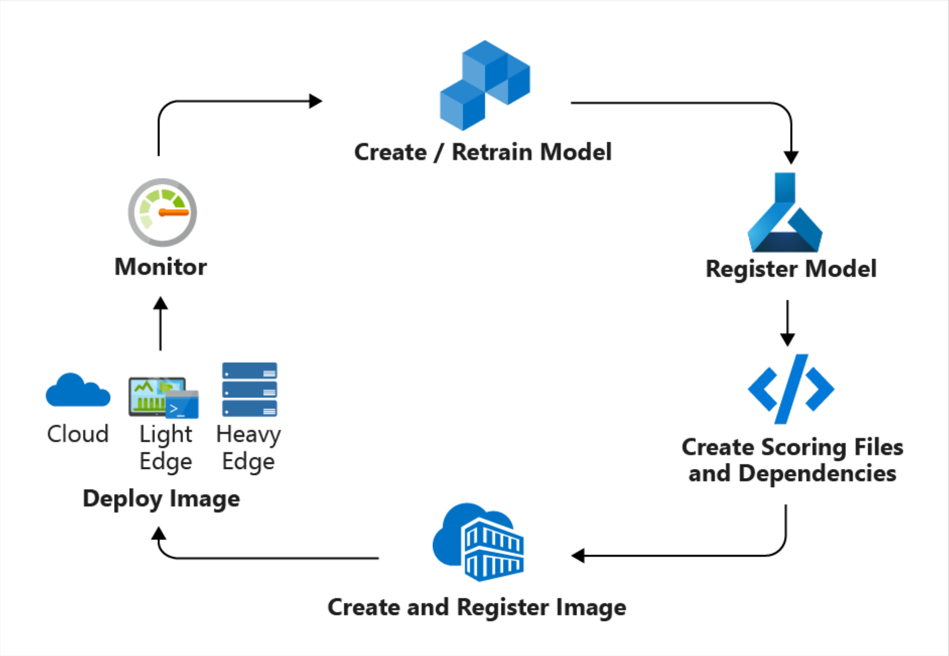
Model Management in Azure ML usually involves these four steps

Step 1: Register Model using the Model Registry

Step 2: Register Image using the Image Registry (the Azure Container Registry)

Step 3: Deploy the Image to cloud or to edge devices

Step 4: Monitor models—you can monitor input, output, and other relevant data from your model.



Azure ML Artifact

Deployment

Deployment is an instantiation of an image

Web service

A deployed web service can run on Azure Container Instances, Azure Kubernetes Service, or field-programmable gate arrays (FPGA).

Can receive scoring requests via an exposed a load-balanced, HTTP endpoint.

Can be monitored by collecting Application Insight telemetry and/or model telemetry.

Azure can automatically scale deployments.

IoT Module

A deployed IoT Module is a Docker container that includes the model, associated script and additional dependencies.

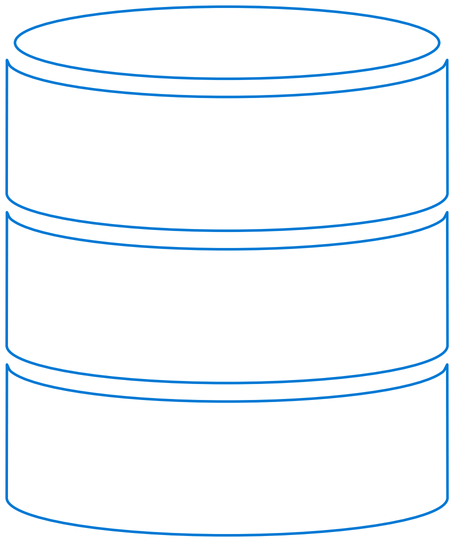
Is deployed using **Azure IoT Edge** on edge devices.

Can be monitored by collecting Application Insight telemetry and/or model telemetry.

Azure IoT Edge will ensure that your module is running and monitor the device that is hosting it.

Azure ML Artifact

Datastore



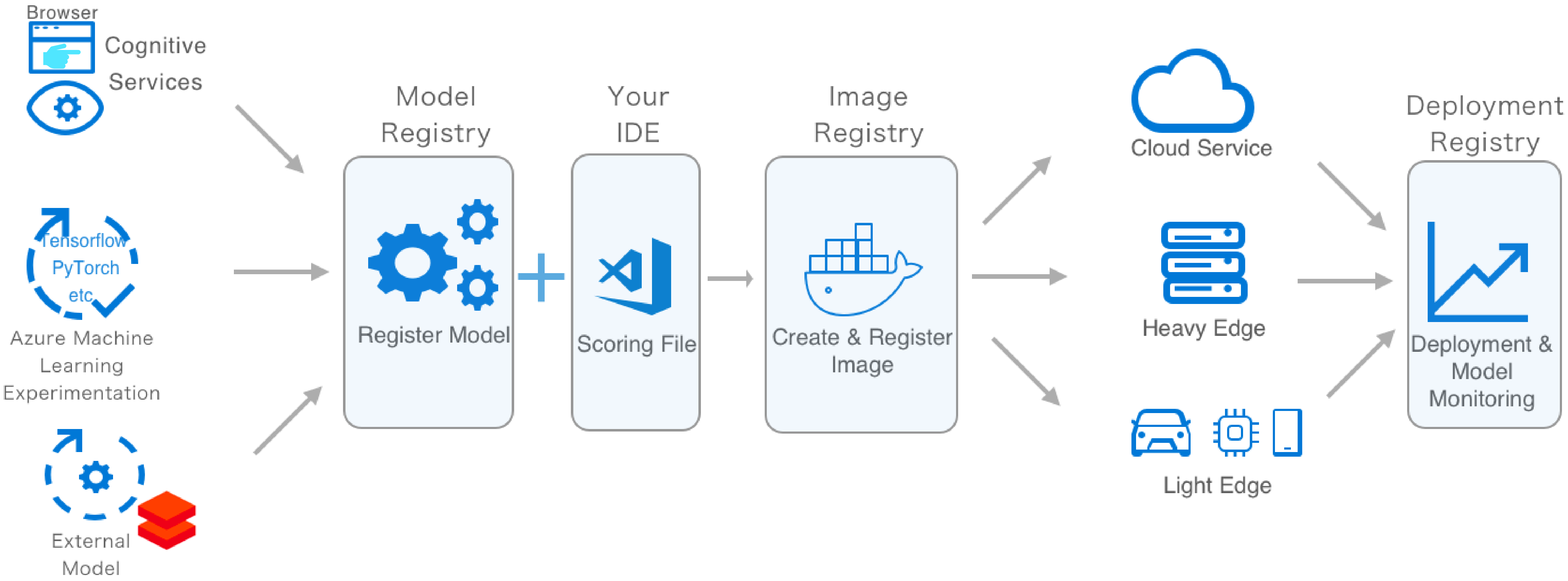
A datastore is a storage abstraction over an Azure Storage Account.

The datastore can use either an Azure blob container or an Azure file share as the backend storage.

Each workspace has a default datastore, and you may register additional datastores.

Use the Python SDK API or Azure Machine Learning CLI to store and retrieve files from the datastore.

Azure ML: How to deploy models at scale

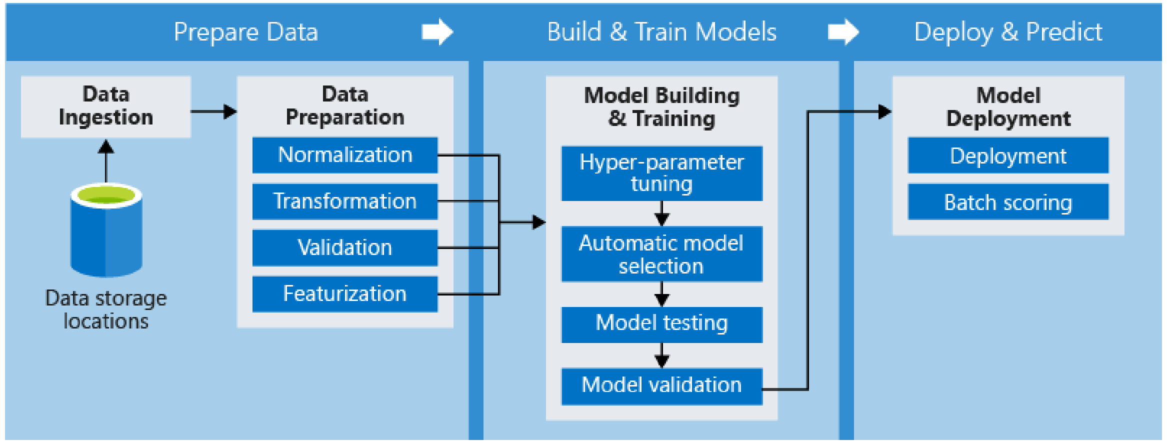


Azure ML Artifact

Pipeline

An Azure ML pipeline consists of a number of steps, where each step can be performed independently or as part of a single deployment command.

A step is a computational unit in the pipeline.
Diagram shows an example pipeline with multiple steps.



Azure ML pipelines enables data scientists, data engineers, and IT professionals to collaborate on the steps involved in: Data preparation, Model training, Model evaluation, Deployment

How pipelines help?

- ✓ Using distinct steps makes it possible to rerun only the steps you need as you tweak and test your workflow.
- ✓ When you rerun a pipeline, the run jumps to the steps that need to be rerun, such as an updated training script, and skips what hasn't changed.
  - ✓ The same holds true for unchanged scripts used for the execution of the step
- ✓ You can use various toolkits and frameworks for each step in your pipeline. Azure coordinates between the various compute targets you use so that your intermediate data can be shared with the downstream compute targets easily.

Azure ML Pipeline

Python SDK

The Azure Machine Learning SDK offers imperative constructs for sequencing and parallelizing the steps in your pipelines when no data dependency is present.

Using declarative data dependencies, you can optimize your tasks.

The SDK includes a framework of pre-built modules for common tasks such as data transfer and model publishing.

The framework can be extended to model your own conventions by implementing custom steps that are reusable across pipelines.

Compute targets and storage resources can also be managed directly from the SDK.

Pipelines can be saved as templates and can be deployed to a REST endpoint so you can schedule batch-scoring or retraining jobs

Azure ML Pipelines

Advantages

Advantage	Description
Unattended runs	Schedule a few steps to run in parallel or in sequence in a reliable and unattended manner. Since data prep and modeling can last days or weeks, you can now focus on other tasks while your pipeline is running.
Mixed and diverse compute	Use multiple pipelines that are reliably coordinated across heterogeneous and scalable computes and storages. Individual pipeline steps can be run on different compute targets, such as HDInsight, GPU Data Science VMs, and Databricks.
Reusability	Pipelines can be templatized for specific scenarios such as retraining and batch scoring. They can be triggered from external systems via simple REST calls.
Tracking and versioning	Instead of manually tracking data and result paths as you iterate, use the pipelines SDK to explicitly name and version your data sources, inputs, and outputs as well as manage scripts and data separately for increased productivity

Azure ML Artifact

Compute Target

Compute Targets are the compute resources used to run training scripts or host your model when deployed as a web service.

They can be created and managed using the Azure Machine Learning SDK or CLI.

You can attach to existing resources.

You can start with local runs on your machine, and then scale up and out to other environments.

Currently supported compute targets

Compute Target	Training	Deployment
Local Computer	✓	
A Linux VM in Azure (such as the Data Science Virtual Machine)	✓	
Azure ML Compute	✓	
Azure Databricks	✓	
Azure Data Lake Analytics	✓	
Apache Spark for HDInsight	✓	
Azure Container Instance		✓
Azure Kubernetes Service		✓
Azure IoT Edge		✓
Field-programmable gate array (FPGA)		✓

Note: it doesn't make sense to train models on IoT edge, for example.

Azure ML

Currently Supported Compute Targets

Compute target	GPU acceleration	Hyperdrive	Automated model selection	Can be used in pipelines
Local computer	Maybe		✓	
Data Science Virtual Machine (DSVM)	✓	✓	✓	✓
Azure ML compute	✓	✓	✓	✓
Azure Databricks	✓		✓	✓
Azure Data Lake Analytics				✓
Azure HDInsight				✓

https://docs.microsoft.com/en-us/azure/machine-learning/service/how-to-set-up-training-targets#supported-compute-targets

Azure Machine Learning

Track experiments and training metrics

Start logging metrics

start_logging - Add logging functions to your training script and start an interactive logging session in the specified experiment. start_logging creates an interactive run for use in scenarios such as notebooks. Any metrics that are logged during the session are added to the run record in the experiment.

```
run = experiment.start_logging()
run.log('alpha', 0.03)
```

ScriptRunConfig - Add logging functions to your training script and load the entire script folder with the run. ScriptRunConfig is a class for setting up configurations for script runs. With this option, you can add monitoring code to be notified of completion or to get a visual widget to monitor.

```
src = ScriptRunConfig(source_directory = './', script = 'train.py', run_config = run_config_user_managed)
run = experiment.submit(src)
```

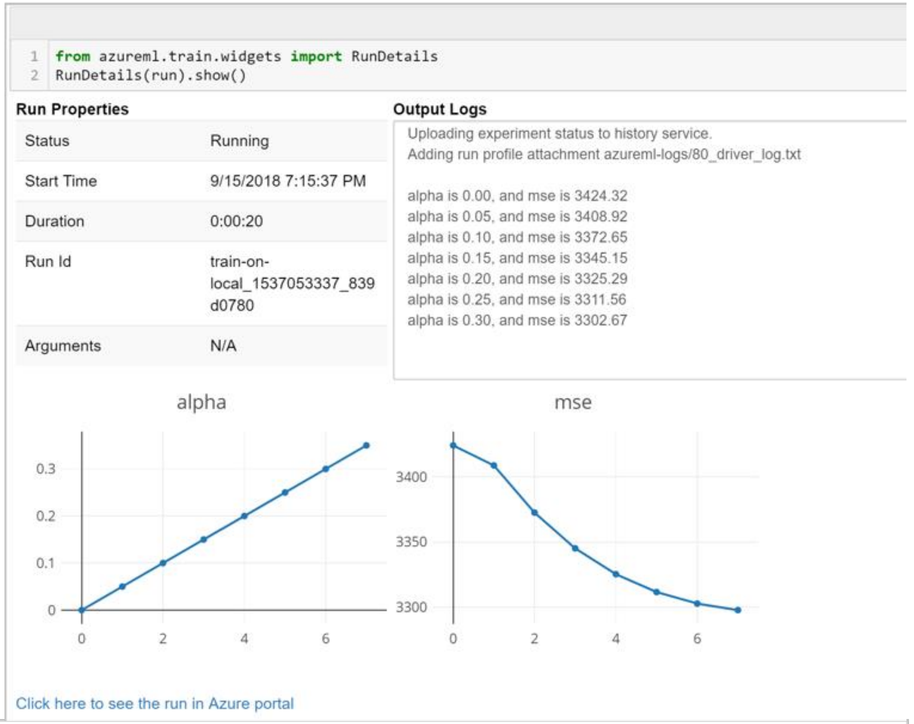
Azure Machine Learning

Track experiments and training metrics

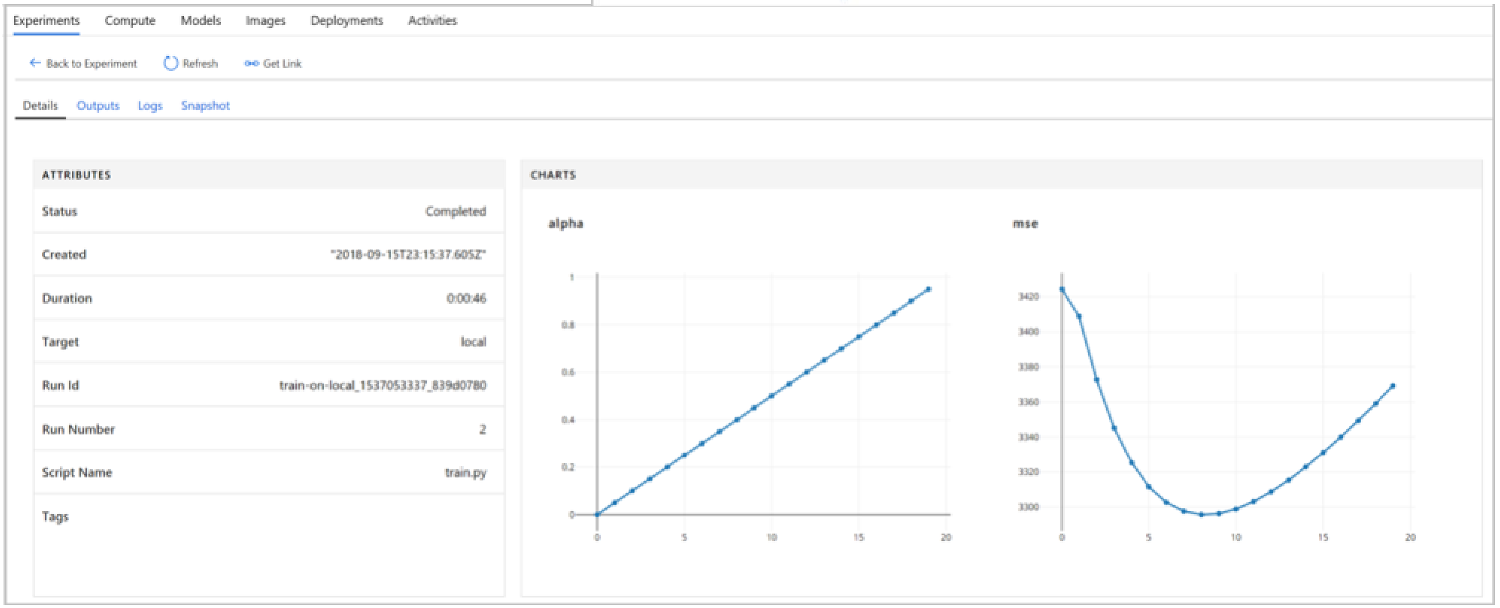
ScriptRunConfig: using ScriptRunConfig method to submit runs, you can watch the progress of the run with a Jupyter notebook widget. Like the run submission, the widget is asynchronous and provides live updates every 10-15 seconds until the job completes.

from azureml.widgets import RunDetails

RunDetails(run).show()



View the experiment in the Azure portal



You can view metrics / loggings for both start_logging and ScriptRunConfig in Azure Portal.

Azure Machine Learning

Data Wrangler – DataPrep SDK: https://docs.microsoft.com/en-us/python/api/azureml-dataprep/?view=azure-dataprep-py

- Automatic file type detection.
- Load from many file types with parsing parameter inference (encoding, separator, headers).
- Type-conversion using inference during file loading
- Connection support for MS SQL Server and Azure Data Lake Storage
- Add column using an expression
- Impute missing values
- Derive column by example
- Filtering
- Custom Python transforms
- Scale through streaming – instead of loading all data in memory
- Summary statistics
- Intelligent time-saving transformations:
    - Fuzzy grouping
    - Derived column by example
    - Automatic split columns by example
    - Impute missing values
    - Automatic join
- Cross-platform functionality with a single code artifact. The SDK also allows for dataflow objects to be serialized and opened in *any* Python environment.

Azure Machine Learning

Azure Machine Learning SDK

```
pip install --upgrade azureml-sdk[notebooks,automl]
```

```
pip install azureml-monitoring
from azureml.monitoring import ModelDataCollector
```

> azureml-monitoring

```
pip install --upgrade
azureml-dataprep
```

import azureml.dataprep as dprep

- azureml.dataprep
- azureml.dataprep.api.builders
- azureml.dataprep.api.expressions
- azureml.dataprep.api.functions

- azureml-core
- azureml-explain-model
- azureml-train-core
- azureml-pipeline-core
- azureml-pipeline-steps
- azureml-train-automl
- azureml-telemetry
- azureml-webservice-schema
- azureml-widgets

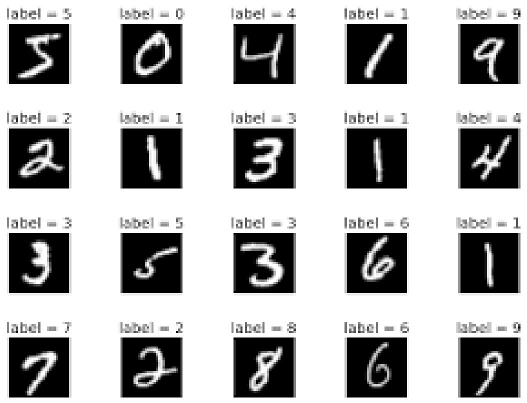
How to use the Azure Machine Learning service: An example using the Python SDK

Setup for Code Example

This example trains a simple logistic regression using the MNIST dataset and scikit-learn with Azure Machine Learning service.

MNIST is a dataset consisting of 70,000 grayscale images.

- Each image is a handwritten digit of 28x28 pixels, representing a number from 0 to 9.
- The goal is to create a multi-class classifier to identify the digit a given image represents.



Step 1 – Create a workspace

```
from azureml.core import Workspace
ws = Workspace.create(name='myworkspace',
                      subscription_id='<azure-subscription-id>',
                      resource_group='myresourcegroup',
                      create_resource_group=True,
                      location='eastus2' # or other supported Azure region
                     )

# see workspace details
ws.get_details()
```

Step 2 – Create an Experiment

Create an experiment to track the runs in the workspace. A workspace can have multiple experiments

```
experiment_name = ‘my-experiment-1'

from azureml.core import Experiment
exp = Experiment(workspace=ws, name=experiment_name)
```

Step 3 – Create remote compute target

```
# choose a name for your cluster, specify min and max nodes
compute_name = os.environ.get("BATCHAI_CLUSTER_NAME", "cpucluster")
compute_min_nodes = os.environ.get("BATCHAI_CLUSTER_MIN_NODES", 0)
compute_max_nodes = os.environ.get("BATCHAI_CLUSTER_MAX_NODES", 4)

# This example uses CPU VM. For using GPU VM, set SKU to STANDARD_NC6
vm_size = os.environ.get("BATCHAI_CLUSTER_SKU", "STANDARD_D2_V2")

provisioning_config = AmlCompute.provisioning_configuration(
                              vm_size = vm_size,
                              min_nodes = compute_min_nodes,
                              max_nodes = compute_max_nodes)
# create the cluster
print(‘ creating a new compute target... ')
compute_target = ComputeTarget.create(ws, compute_name, provisioning_config)

# You can poll for a minimum number of nodes and for a specific timeout.
# if no min node count is provided it will use the scale settings for the cluster
compute_target.wait_for_completion(show_output=True,
                                   min_node_count=None, timeout_in_minutes=20)
```

Zero is the default. If min is zero then the cluster is automatically deleted when no jobs are running on it.

Step 4 – Upload data to the cloud

First load the compressed files into numpy arrays. Note the '*load_data*' is a custom function that simply parses the compressed files into numpy arrays.

```
# note that while loading, we are shrinking the intensity values (X) from 0-255 to 0-1 so that the model converge faster.
X_train = load_data('./data/train-images.gz', False) / 255.0
y_train = load_data('./data/train-labels.gz', True).reshape(-1)

X_test = load_data('./data/test-images.gz', False) / 255.0
y_test = load_data('./data/test-labels.gz', True).reshape(-1)
```

Now make the data accessible remotely by uploading that data from your local machine into Azure so it can be accessed for remote training. The files are uploaded into a directory named mnist at the root of the datastore.

```
ds = ws.get_default_datastore()
print(ds.datastore_type, ds.account_name, ds.container_name)

ds.upload(src_dir='./data', target_path='mnist', overwrite=True, show_progress=True)
```

We now have everything you need to start training a model.

Step 5 – Train a local model

Train a simple logistic regression model using scikit-learn locally. This should take a minute or two.

```
%%time from sklearn.linear_model import LogisticRegression
clf = LogisticRegression()
clf.fit(X_train, y_train)

# Next, make predictions using the test set and calculate the accuracy
y_hat = clf.predict(X_test)
print(np.average(y_hat == y_test))
```

You should see the local model accuracy displayed. [It should be a number like 0.915]

Step 6 – Train model on remote cluster

To submit a training job to a remote you have to perform the following tasks:

- 6.1: Create a directory
- 6.2: Create a training script
- 6.3: Create an estimator object
- 6.4: Submit the job

Step 6.1 – Create a directory

Create a directory to deliver the required code from your computer to the remote resource.

```
import os
script_folder = './sklearn-mnist' os.makedirs(script_folder, exist_ok=True)
```

Step 6.2 – Create a Training Script (1/2)

```
%%writefile $script_folder/train.py
# load train and test set into numpy arrays
# Note: we scale the pixel intensity values to 0-1 (by dividing it with 255.0) so the model can
# converge faster.
# ‘data_folder’ variable holds the location of the data files (from datastore)
Reg = 0.8 # regularization rate of the logistic regression model.
X_train = load_data(os.path.join(data_folder, 'train-images.gz'), False) / 255.0
X_test  = load_data(os.path.join(data_folder, 'test-images.gz'), False) / 255.0
y_train = load_data(os.path.join(data_folder, 'train-labels.gz'), True).reshape(-1)
y_test  = load_data(os.path.join(data_folder, 'test-labels.gz'), True).reshape(-1)
print(X_train.shape, y_train.shape, X_test.shape, y_test.shape, sep = '\n’)

# get hold of the current run
run = Run.get_context()
#Train a logistic regression model with regularizaion rate of’ ‘reg’
clf = LogisticRegression(C=1.0/reg, random_state=42)
clf.fit(X_train, y_train)
```

Step 6.2 – Create a Training Script (2/2)

```
print('Predict the test set’)
y_hat = clf.predict(X_test)

# calculate accuracy on the prediction
acc = np.average(y_hat == y_test)
print('Accuracy is', acc)


run.log('regularization rate', np.float(args.reg))
run.log('accuracy', np.float(acc)) os.makedirs('outputs', exist_ok=True)


# The training script saves the model into a directory named ‘outputs’. Note files saved in the
# outputs folder are automatically uploaded into experiment record. Anything written in this
# directory is automatically uploaded into the workspace.
joblib.dump(value=clf, filename='outputs/sklearn_mnist_model.pkl')
```

The directory that contains the scripts. All the files in this directory are uploaded into the cluster nodes for execution

Step 6.3 – Create an Estimator

An estimator object is used to submit the run.

```
from azureml.train.estimator import Estimator

script_params = { '--data-folder': ds.as_mount(), '--regularization': 0.8 }

est = Estimator(source_directory=script_folder,
                script_params=script_params,
                compute_target=compute_target,
                entry_script='train.py’,
                conda_packages=['scikit-learn'])
```

Name of estimator

Python Packages needed for training

Training Script Name

Compute target (Batch AI in this case)

Parameters required from the training script

Step 6.4 – Submit the job to the cluster for training

```
run = exp.submit(config=est)
```

What happens after you submit the job?

Image creation

A Docker image is created matching the Python environment specified by the estimator. The image is uploaded to the workspace. Image creation and uploading takes about 5 minutes.

This happens once for each Python environment since the container is cached for subsequent runs. During image creation, logs are streamed to the run history. You can monitor the image creation progress using these logs.

Scaling

If the remote cluster requires more nodes to execute the run than currently available, additional nodes are added automatically. Scaling typically takes about 5 minutes.

Running

In this stage, the necessary scripts and files are sent to the compute target, then data stores are mounted/copied, then the entry_script is run. While the job is running, stdout and the ./logs directory are streamed to the run history. You can monitor the run's progress using these logs.

Post-Processing

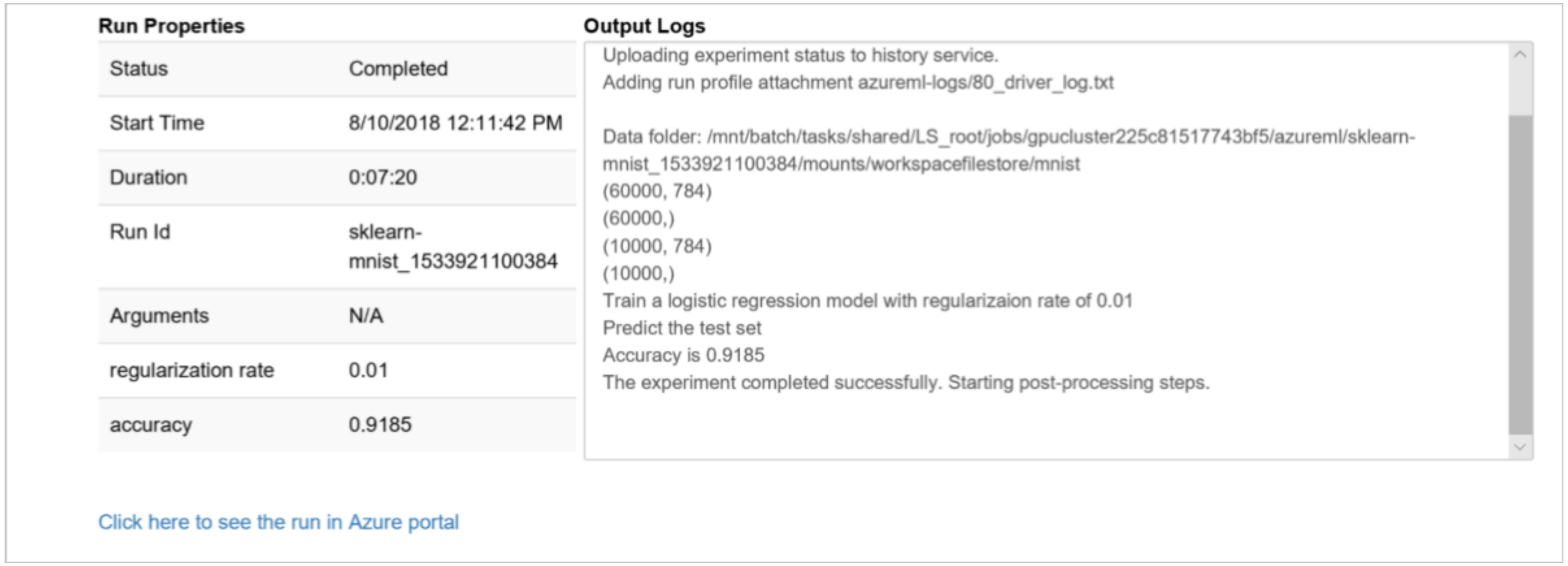
The ./outputs directory of the run is copied over to the run history in your workspace so you can access these results.

Step 7 – Monitor a run

You can watch the progress of the run with a Jupyter widget. The widget is asynchronous and provides live updates every 10-15 seconds until the job completes.

```python
from azureml.widgets import RunDetails
RunDetails(run).show()
```

Here is a still snapshot of the widget shown at the end of training:



Step 8 – See the results

As model training and monitoring happen in the background. Wait until the model has completed training before running more code. Use *wait_for_completion* to show when the model training is complete

Specify 'True' for a verbose log

```
run.wait_for_completion(show_output=False)

# now there is a trained model on the remote cluster
print(run.get_metrics())
```

Displays the accuracy of the model. You should see an output that looks like this.

```
{'regularization rate': 0.8, 'accuracy': 0.9204}
```

Step 9 – Register the model

Recall that the last step in the training script is:

```
joblib.dump(value=clf, filename='outputs/sklearn_mnist_model.pkl')
```

This wrote the file 'outputs/sklearn_mnist_model.pkl' in a directory named 'outputs' in the VM of the cluster where the job is executed.

- outputs is a special directory in that all content in this directory is automatically uploaded to your workspace.
- This content appears in the run record in the experiment under your workspace.
- Hence, the model file is now also available in your workspace.

```
# register the model in the workspace
model = run.register_model (
                             model_name='sklearn_mnist’,
                             model_path='outputs/sklearn_mnist_model.pkl’)
```

The model is now available to query, examine, or deploy

Step 9 – Deploy the Model

Deploy the model registered in the previous slide, to Azure Container Instance (ACI) as a Web Service

There are 4 steps involved in model deployment

Step 9.1 – Create scoring script

Step 9.2 – Create environment file

Step 9.3 – Create configuration file

Step 9.4 – Deploy to ACI!

Step 9.1 – Create the scoring script

Create the scoring script, called score.py, used by the web service call to show how to use the model. It requires two functions – init() and run (input data)

The init() function, typically loads the model into a global object. This function is run only once when the Docker container is started.

```
from azureml.core.model import Model

def init():
      global model
      # retreive the path to the model file using the model name
      model_path = Model.get_model_path('sklearn_mnist’)
      model = joblib.load(model_path)

def run(raw_data):
      data = np.array(json.loads(raw_data)['data’])
      # make prediction
      y_hat = model.predict(data)
      return json.dumps(y_hat.tolist())
```

The run(input_data) function uses the model to predict a value based on the input data. Inputs and outputs to the run typically use JSON for serialization and de-serialization, but other formats are supported

Step 9.2 – Create environment file

Create an environment file, called *myenv.yml*, that specifies all of the script's package dependencies. This file is used to ensure that all of those dependencies are installed in the Docker image. This example needs scikit-learn and azureml-sdk.

```
from azureml.core.conda_dependencies import CondaDependencies

myenv = CondaDependencies()
myenv.add_conda_package("scikit-learn")

with open("myenv.yml","w") as f:
        f.write(myenv.serialize_to_string())
```

Step 9.3 – Create configuration file

Create a deployment configuration file and specify the number of CPUs and gigabyte of RAM needed for the ACI container. Here we will use the defaults (1 core and 1 gigabyte of RAM)

```
from azureml.core.webservice import AciWebservice

aciconfig = AciWebservice.deploy_configuration(cpu_cores=1, memory_gb=1,
                                               tags={"data": "MNIST", "method" : "sklearn"},
                                               description='Predict MNIST with sklearn')
```

Step 9.4 – Deploy the model to ACI

```
%%time
from azureml.core.webservice import Webservice
from azureml.core.image import ContainerImage

# configure the image
image_config = ContainerImage.image_configuration(
                                        execution_script ="score.py",
                                        runtime ="python",
                                        conda_file ="myenv.yml")

service = Webservice.deploy_from_model(workspace=ws, name='sklearn-mnist-svc’,
                                        deployment_config=aciconfig, models=[model],
                                        image_config=image_config)

service.wait_for_deployment(show_output=True)
```

Build an image using:

- The scoring file (score.py)
- The environment file (myenv.yml)
- The model file

Register that image under the workspace and send the image to the ACI container.

Start up a container in ACI using the image

Step 10 – Test the deployed model using the HTTP end point

Test the deployed model by sending images to be classified to the HTTP endpoint

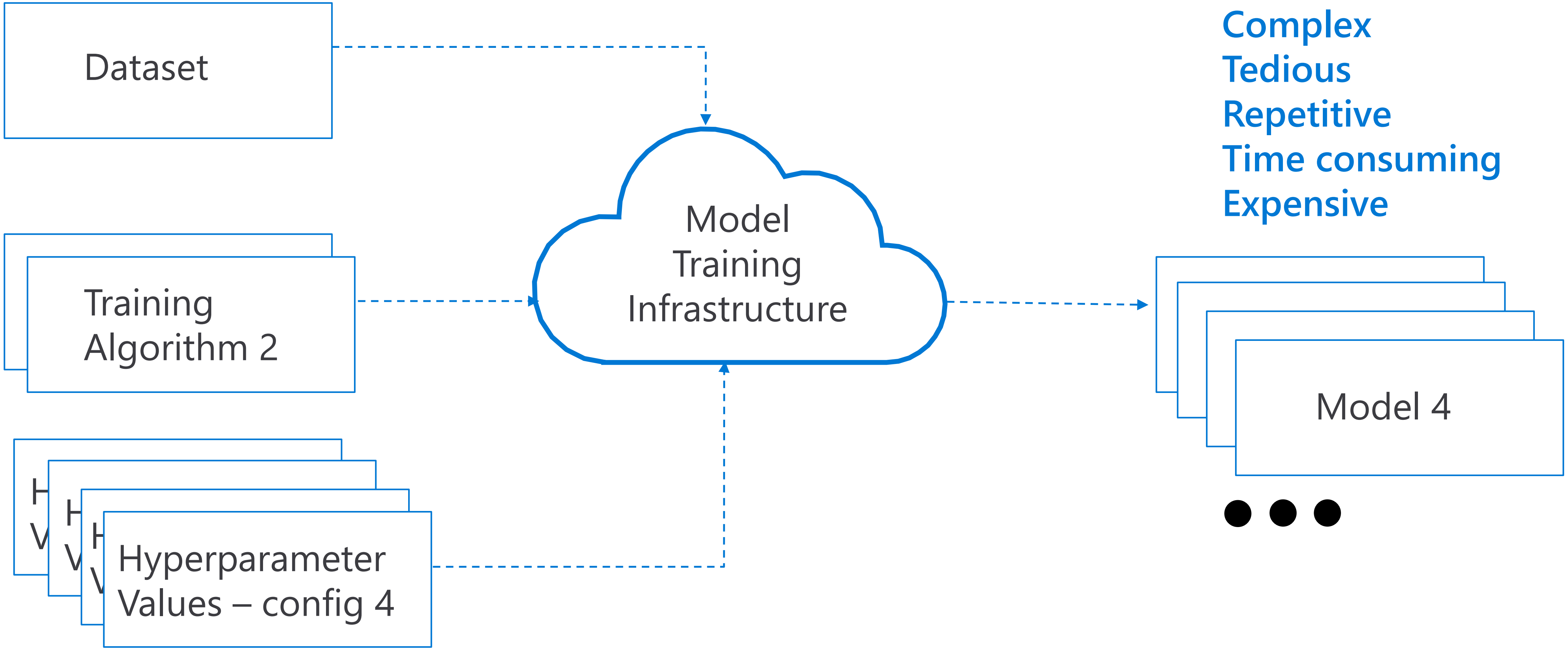
```
import requests
import json

# send a random row from the test set to score
random_index = np.random.randint(0, len(X_test)-1)
input_data = "{\"data\": [" + str(list(X_test[random_index])) + "]}"

headers = {'Content-Type':'application/json’}

resp = requests.post(service.scoring_uri, input_data, headers=headers)

print("POST to url", service.scoring_uri)
#print("input data:", input_data)
print("label:", y_test[random_index])
print("prediction:", resp.text)
```

Send the data to the HTTP end-point for scoring

https://github.com/Azure/MachineLearningNotebooks/tree/master/tutorials
https://docs.microsoft.com/en-us/azure/machine-learning/service/tutorial-train-models-with-aml

Azure Automated Machine Learning ‘simplifies’ the creation and selection of the optimal model

Typical 'manual' approach to hyperparameter tuning



Complex
Tedious
Repetitive
Time consuming
Expensive

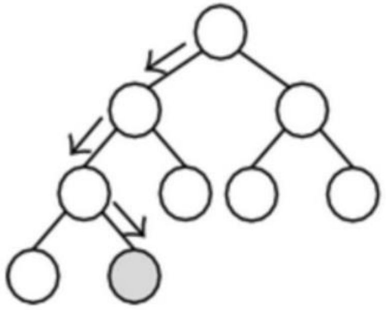
What are Hyperparameters?

Adjustable parameters that govern model training

Chosen prior to training, stay constant during training

Model performance heavily depends on hyperparameter

Number of leaves	Minimum leaf instances	Learning rate	Number of trees
8	10	0.1	500
8	1	0.05	500
8	1	0.2	100
32	1	0.05	100
8	10	0.2	100
32	1	0.025	500
8	10	0.05	500
32	1	0.1	100
8	1	0.025	500
8	50	0.05	500
32	10	0.025	500
8	50	0.025	500
32	10	0.05	100
8	10	0.025	500
32	10	0.2	20
8	1	0.1	500
32	10	0.1	100
8	1	0.1	100
8	10	0.1	100



Setting
Number Of Leaves
Minimum Leaf Instances
Learning Rate
Number Of Trees

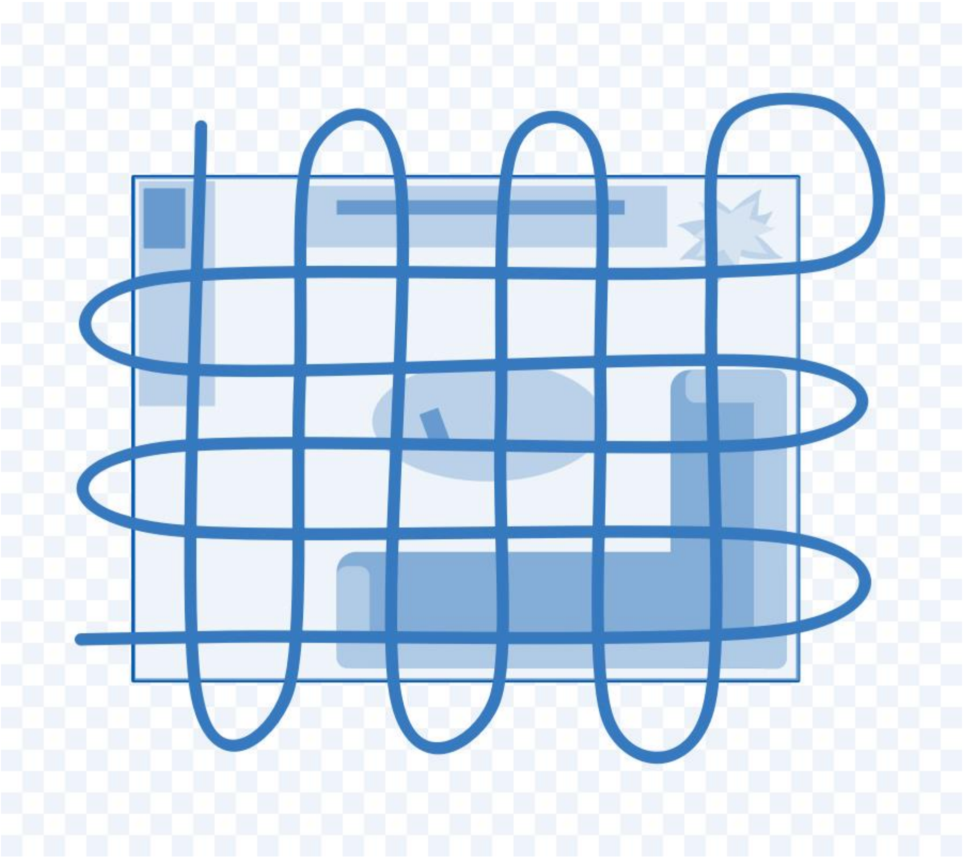
Challenges with Hyperparameter Selection

The search space to explore—i.e. evaluating all possible combinations—is huge.

Sparsity of good configurations.
Very few of all possible configurations are optimal.

Evaluating each configuration is resource and time consuming.

Time and resources are limited.



Azure Automated ML: Sampling to generate new runs

HyperDrive

Define hyperparameter search space

```
{
    "learning_rate": uniform(0, 1),
    "num_layers": choice(2, 4, 8)
    …
}
```

Sampling algorithm

```
Config1= {"learning_rate": 0.2,
"num_layers": 2, …}

Config2= {"learning_rate": 0.5,
"num_layers": 4, …}

Config3= {"learning_rate": 0.9,
"num_layers": 8, …}

…
```

Supported sampling algorithms:
Grid Sampling
Random Sampling
Bayesian Optimization

Azure Automated ML

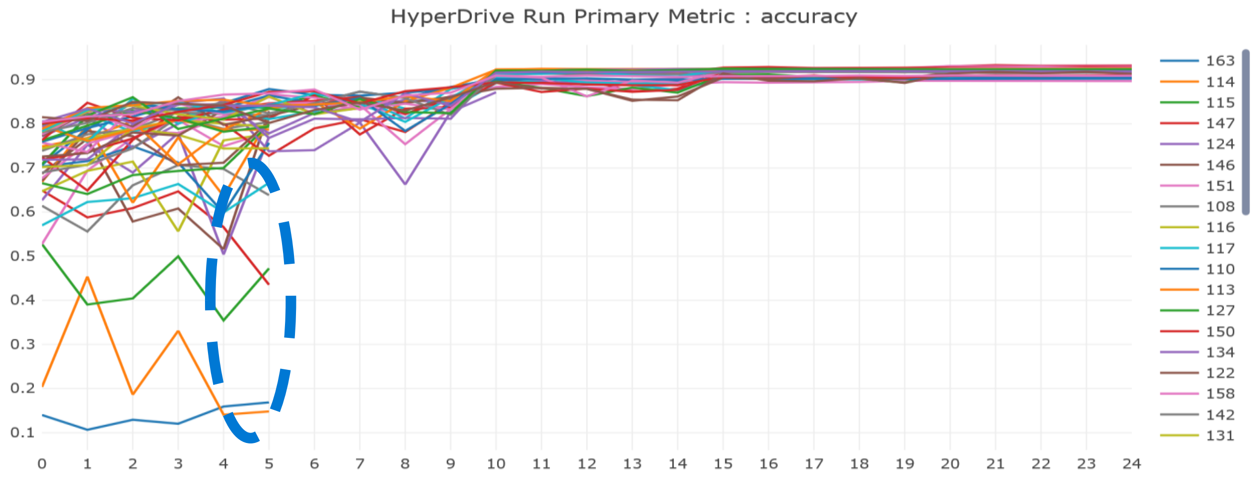
HyperDrive

Evaluate training runs for specified primary metric

Use resources to explore new configurations

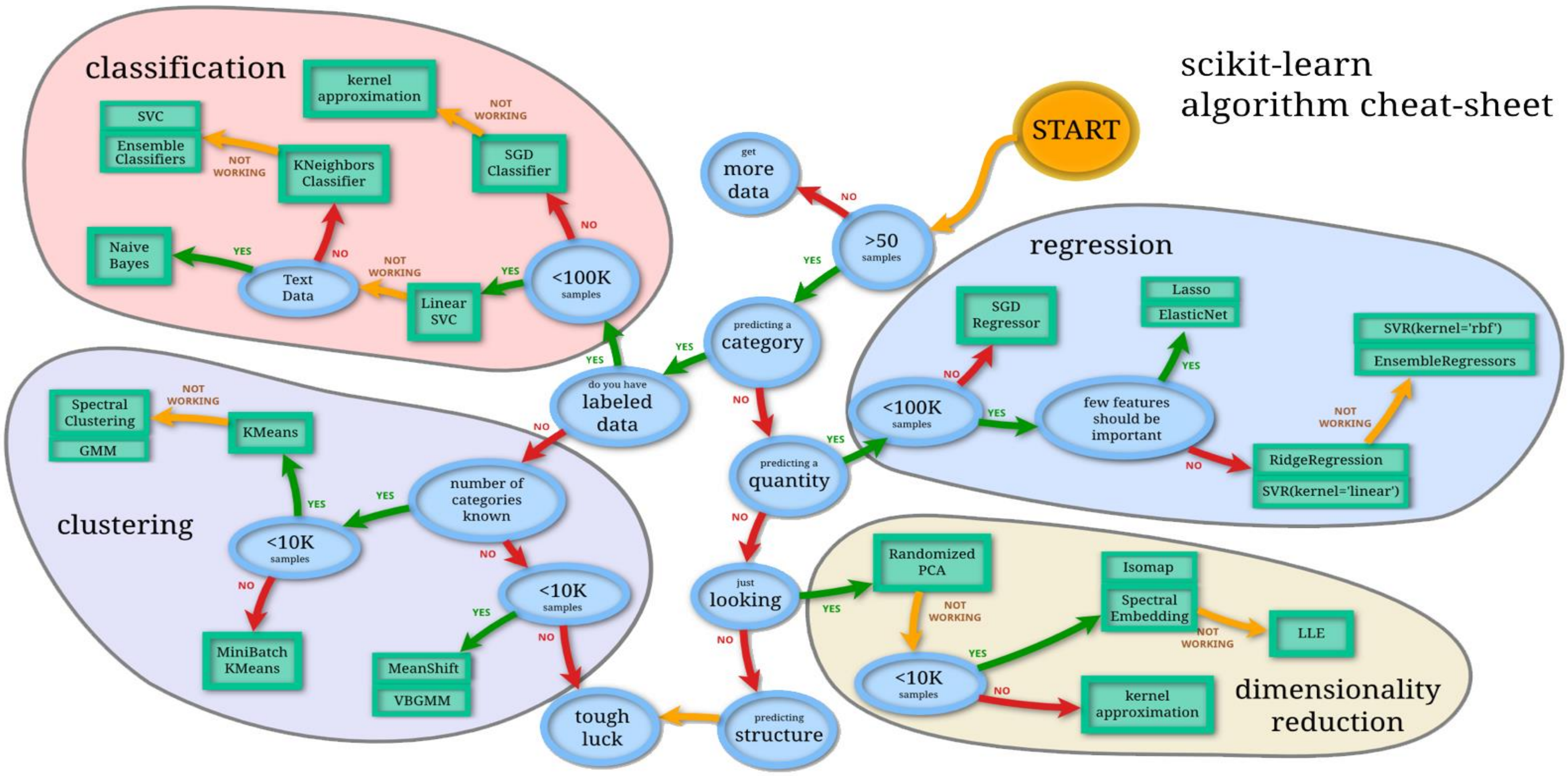
Early terminate poor performing training runs. Early termination policies include:

- Bandit policy
- Median Stopping policy
- Truncation Selection policy



- Define the parameter search space
- Specify a primary metric to optimize
- Specify early termination criteria for poorly performing runs
- Allocate resources for hyperparameter tuning
- Launch an experiment with the above configuration
- Visualize the training runs
- Select the best performing configuration for your model

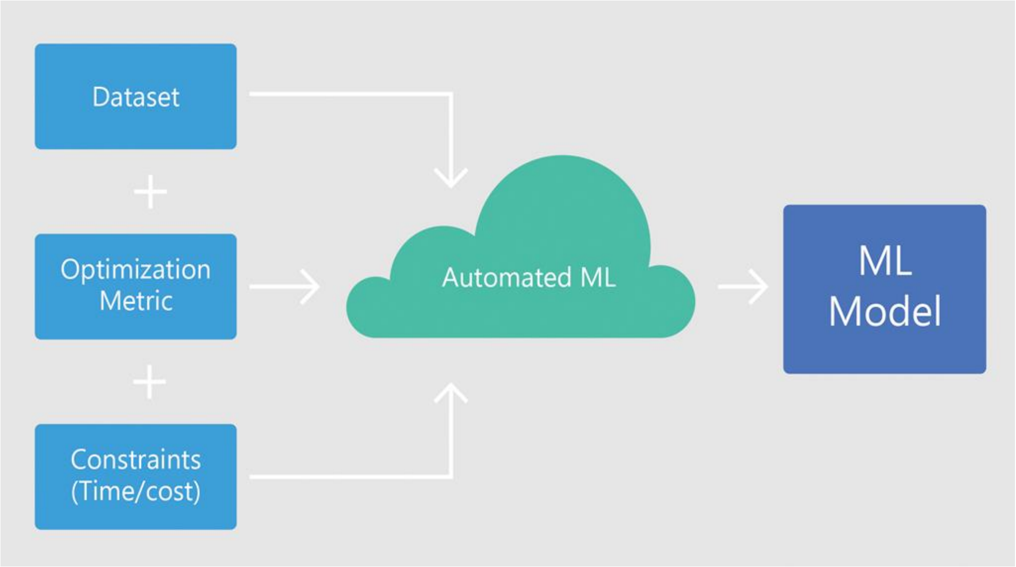
Complexity of Machine Learning

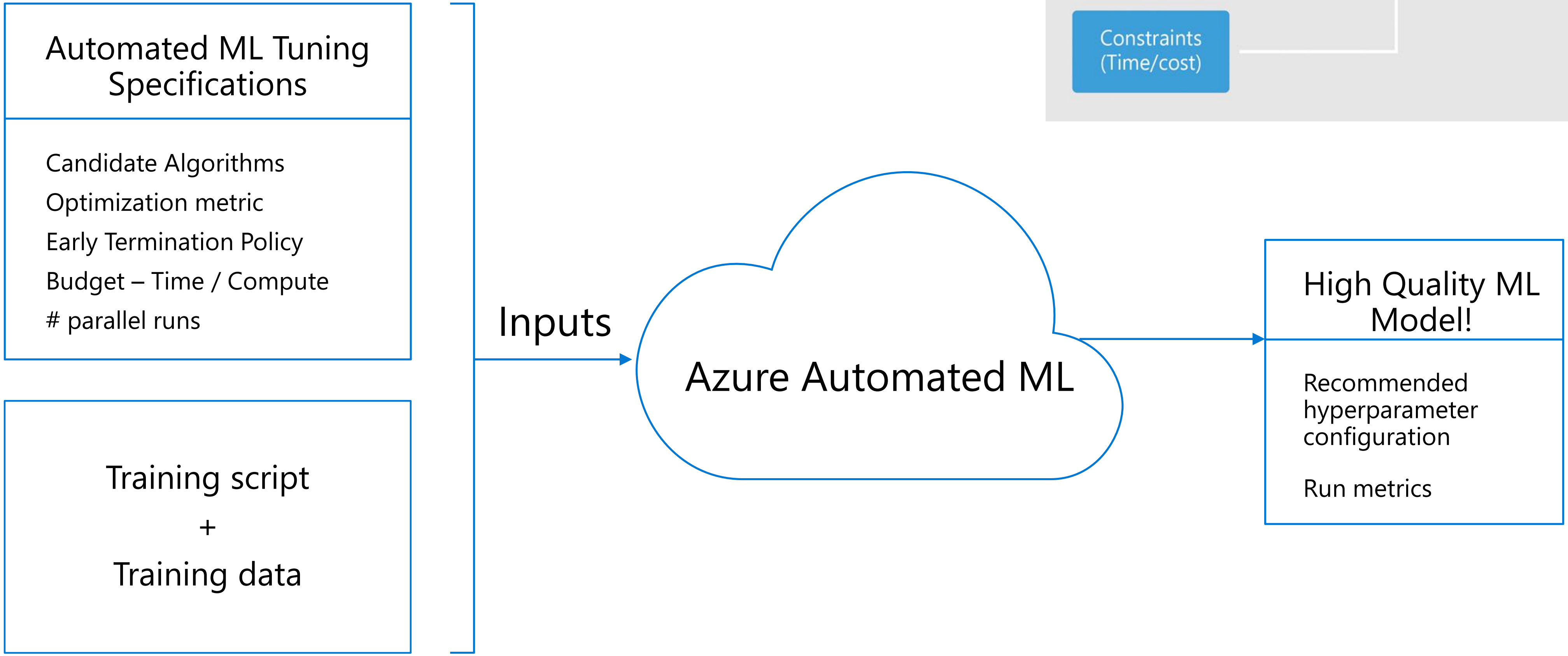


Source: http://scikit-learn.org/stable/tutorial/machine_learning_map/index.html

Azure Automated ML

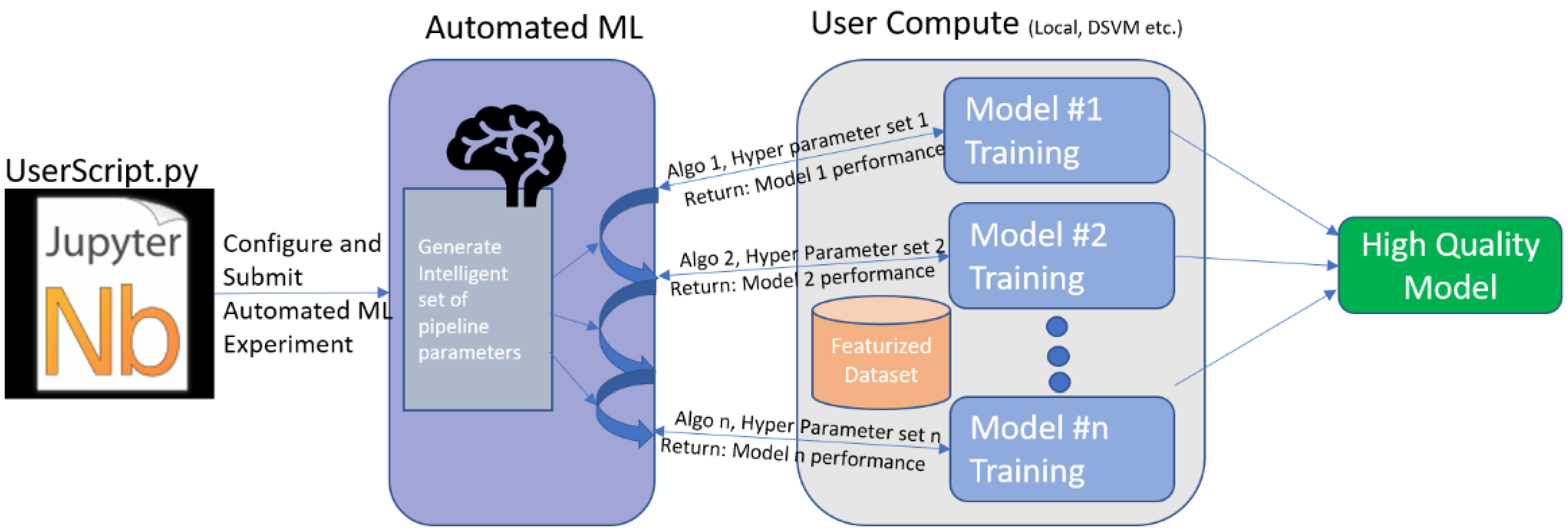
Conceptual Overview





Azure Automated ML

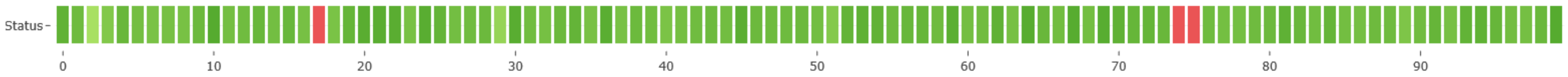
How It Works



During training, the Azure Machine Learning service creates a number of pipelines that try different algorithms and parameters. It will stop once it hits the iteration limit you provide, or when it reaches the target value for the metric you specify.

Azure Automated ML – Sample Output

AutoML_ab755820-4bfd-4e8a-8b4b-9e0a2446b1c2:
Status: Completed

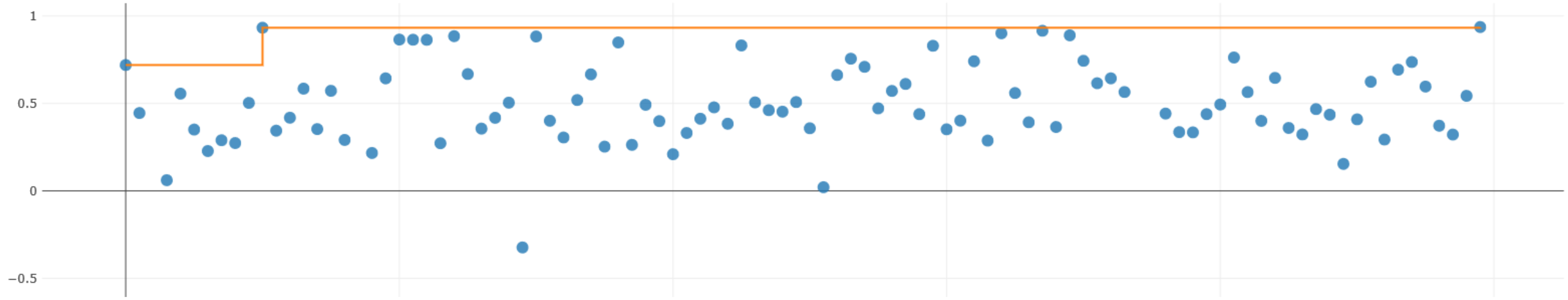


Iteration	Pipeline	Iteration metric	Best metric	Status	Duration	Started	Run Id
99	Ensemble	0.93702349	0.93702349	Completed	0:02:18	Dec 4, 2018 12:18 AM	
10	MaxAbsScaler, LightGBM	0.93289307	0.93289307	Completed	0:01:22	Dec 3, 2018 7:49 PM	
67	SparseNormalizer, LightGBM	0.9154763	0.93289307	Completed	0:01:31	Dec 3, 2018 10:19 PM	
64	MaxAbsScaler, LightGBM	0.90148724	0.93289307	Completed	0:01:24	Dec 3, 2018 10:09 PM	
69	MaxAbsScaler, LightGBM	0.88975241	0.93289307	Completed	0:00:55	Dec 3, 2018 10:22 PM	

Pages: 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 ... Next Last 5 per page

r2_score

AutoML Run with metric : r2_score



Azure Automated ML

Use via the Python SDK

jupyter 102.auto-ml-regression (unsaved changes)

File Edit View Insert Cell Kernel Widgets Help Not Trusted Python

Run Code

Instantiate Auto ML Regressor

Instantiate a AutoML Object This creates an Experiment in Azure ML. You can reuse this objects to trigger multiple runs. Each run will be part of the same experiment.

Property	Description
primary_metric	This is the metric that you want to optimize. Auto ML Regressor supports the following primary metrics *spearman_correlation* *normalized_root_mean_squared_error* *r2_score*
max_time_sec	Time limit in seconds for each iterations
iterations	Number of iterations. In each iteration Auto ML Classifier trains the data with a specific pipeline
num_cross_folds	Cross Validation split

In [5]:

```
from azureml.train.automl import AutoMLConfig

automl_config = AutoMLConfig(task = 'regression',
                             debug_log = 'automl_errors.log',
                             primary_metric = 'spearman_correlation',
                             max_time_sec = 12000,
                             iterations = 10,
                             n_cross_validations = 3,
                             verbosity = logging.INFO,
                             X = X,
                             y = y,
                             path=project_folder)
```

Training the Model

You can call the fit method on the AutoML instance and pass the run configuration. For Local runs the execution is synchronous. Depending on the data and number of iterations this can run for while. You will see the currently running iterations printing to the console.

fit method on Auto ML Regressor triggers the training of the model. It can be called with the following parameters

Parameter	Description
X	(sparse) array-like, shape = [n_samples, n_features]
y	(sparse) array-like, shape = [n_samples,], [n_samples, n_classes] Multi-class targets. An indicator matrix turns on multilabel classification.
compute_target	Indicates the compute used for training. *local* indicates train on the same compute which hosts the jupyter notebook. For DSVM and Batch AI please refer to the relevant notebooks.
show_output	True/False to turn on/off console output

In [6]:

```
local_run = experiment.submit(automl_config, show_output=True)
```

```
Parent Run ID: AutoML_e7a4236e-8935-4e93-888d-1ea8310a6b22
*******************************************************************************************
ITERATION: The iteration being evaluated.
PIPELINE: A summary description of the pipeline being evaluated.
DURATION: Time taken for the current iteration.
METRIC: The result of computing score on the fitted pipeline.
BEST: The best observed score thus far.
*******************************************************************************************

 ITERATION   PIPELINE                                  DURATION         METRIC    BEST
         0    Normalize extra trees regressor          0:00:12.069893    0.688   0.688
         1    Normalize lightGBM regressor             0:00:11.192919    0.597   0.688
         2    Normalize Elastic net                    0:00:09.866233    0.689   0.689
         3    Scale 0/1 lightGBM regressor             0:00:10.069764    0.656   0.689
         4    Robust Scaler kNN regressor              0:00:09.090668    0.598   0.689
         5    Normalize lightGBM regressor             0:00:12.562876    0.649   0.689
         6    Robust Scaler kNN regressor              0:00:09.361137    0.600   0.689
         7    Normalize SGD regressor                  0:00:09.010672    0.070   0.689
         8    Scale 0/1 extra trees regressor          0:00:10.442752    0.685   0.689
         9    Robust Scaler Gradient boosting regres0:00:09.567582    0.651   0.689
```

https://docs.microsoft.com/en-us/python/api/azureml-train-automl/azureml.train.automl.automlexplainer?view=azure-ml-py

Azure Automated ML

Current Capabilities

Category		Value
ML Problem Spaces		Classification Regression Forecasting
Frameworks		Scikit Learn
Languages		Python
Data Type and Data Formats		Numerical Text Scikit-learn supported data formats (Numpy, Pandas)
Data sources		Local Files, Azure Blob Storage
Compute Target	Automated Hyperparameter Tuning	Azure ML Compute (Batch AI), Azure Databricks
	Automated Model Selection	Local Compute, Azure ML Compute (Batch AI), Azure Databricks

Azure Automated ML

Algorithms Currently Supported

Classification	Regression	Forecasting
Logistic Regression	Elastic Net	Elastic Net
Stochastic Gradient Descent (SGD)	Light GBM	Light GBM
Naive Bayes	Gradient Boosting	Gradient Boosting
C-Support Vector Classification (SVC)	Decision Tree	Decision Tree
Linear SVC	K Nearest Neighbors	K Nearest Neighbors
K Nearest Neighbors	LARS Lasso	LARS Lasso
Decision Tree	Stochastic Gradient Descent (SGD)	Stochastic Gradient Descent (SGD)
Random Forest	Random Forest	Random Forest
Extremely Randomized Trees	Extremely Randomized Trees	Extremely Randomized Trees
Gradient Boosting		
Light GBM		

Property	Description	Default Value
task	Specify the type of machine learning problem. Allowed values are Classification Regression Forecasting	None
primary_metric	Metric that you want to optimize in building your model. For example, if you specify accuracy as the primary_metric, automated machine learning looks to find a model with maximum accuracy. You can only specify one primary_metric per experiment. Allowed values are **Classification**: accuracy AUC_weighted precision_score_weighted balanced_accuracy average_precision_score_weighted **Regression**: normalized_mean_absolute_error spearman_correlation normalized_root_mean_squared_error normalized_root_mean_squared_log_error R2_score	For Classification: accuracy For Regression: spearman_correlati on
experiment_exit_score	You can set a target value for your primary_metric. Once a model is found that meets the primary_metric target, automated machine learning will stop iterating and the experiment terminates. If this value is not set (default), Automated machine learning experiment will continue to run the number of iterations specified in iterations. Takes a double value. If the target never reaches, then Automated machine learning will continue until it reaches the number of iterations specified in iterations.	None
iterations	Maximum number of iterations. Each iteration is equal to a training job that results in a pipeline. Pipeline is data preprocessing and model. To get a high-quality model, use 250 or more	100
max_concurrent_iterations	Max number of iterations to run in parallel. This setting works only for remote compute.	1
max_cores_per_iteration	Indicates how many cores on the compute target would be used to train a single pipeline. If the algorithm can leverage multiple cores, then this increases the performance on a multi-core machine. You can set it to -1 to use all the cores available on the machine.	1
iteration_timeout_minutes	Limits the amount of time (minutes) a particular iteration takes. If an iteration exceeds the specified amount, that iteration gets canceled. If not set, then the iteration continues to run until it is finished.	None
n_cross_validations	Number of cross validation splits	None
validation_size	Size of validation set as percentage of all training sample.	None
preprocess	True/False True enables experiment to perform preprocessing on the input. Following is a subset of preprocessingMissing Data: Imputes the missing data- Numerical with Average, Text with most occurrence Categorical Values: If data type is numeric and number of unique values is less than 5 percent, Converts into one-hot encoding Etc. for complete list check [the GitHub repository] Note : if data is sparse you cannot use preprocess = true	False
blacklist_models	Automated machine learning experiment has many different algorithms that it tries. Configure to exclude certain algorithms from the experiment. Useful if you are aware that algorithm(s) do not work well for your dataset. Excluding algorithms can save you compute resources and training time. Allowed values for Classification LogisticRegressionSGDMultinomialNaiveBayesBernoulliNaiveBayesSVMLinearSVMKNNDecisionTreeRandomForestExtremeRandomTreesLightGBMGradientBoostingTensorFlowDNNTensorFlowLinearClassifier Allowed values for Regression ElasticNetGradientBoostingDecisionTreeKNNLassoLarsSGD RandomForestExtremeRandomTreeLightGBMTensorFlowLinearRegressorTensorFlowDNN Allowed values for Forecasting ElasticNetGradientBoostingDecisionTreeKNNLassoLarsSGD RandomForestExtremeRandomTreeLightGBMTensorFlowLinearRegressorTensorFlowDNN	None
whitelist_models	Automated machine learning experiment has many different algorithms that it tries. Configure to include certain algorithms for the experiment. Useful if you are aware that algorithm(s) do work well for your dataset. Allowed values for Classification LogisticRegressionSGDMultinomialNaiveBayesBernoulliNaiveBayesSVMLinearSVMKNNDecisionTreeRandomForestExtremeRandomTreesLightGBMGradientBoostingTensorFlowDNNTensorFlowLinearClassifier Allowed values for Regression ElasticNetGradientBoostingDecisionTreeKNNLassoLarsSGD RandomForestExtremeRandomTreeLightGBMTensorFlowLinearRegressorTensorFlowDNN Allowed values for Forecasting ElasticNetGradientBoostingDecisionTreeKNNLassoLarsSGD RandomForestExtremeRandomTreeLightGBMTensorFlowLinearRegressorTensorFlowDNN	None
verbosity	Controls the level of logging with INFO being the most verbose and CRITICAL being the least. Verbosity level takes the same values as defined in the python logging package. Allowed values are: logging.INFOlogging.WARNINGlogging.ERRORlogging.CRITICAL	logging.INFO
X	All features to train with	None
y	Label data to train with. For classification, should be an array of integers.	None
X_valid	*Optional* All features to validate with. If not specified, X is split between train and validate	None
y_valid	*Optional* The label data to validate with. If not specified, y is split between train and validate	None
sample_weight	*Optional* A weight value for each sample. Use when you would like to assign different weights for your data points	None
sample_weight_valid	*Optional* A weight value for each validation sample. If not specified, sample_weight is split between train and validate	None
run_configuration	RunConfiguration object. Used for remote runs.	None
data_script	Path to a file containing the get_data method. Required for remote runs.	None
model_explainability	*Optional* True/False True enables experiment to perform feature importance for every iteration. You can also use explain_model() method on a specific iteration to enable feature importance on-demand for that iteration after experiment is complete.	False
enable_ensembling	Flag to enable an ensembling iteration after all the other iterations complete.	True
ensemble_iterations	Number of iterations during which we choose a fitted pipeline to be part of the final ensemble.	15
experiment_timeout_minutes	Limits the amount of time (minues) that the whole experiment run can take	None

Azure Automated ML

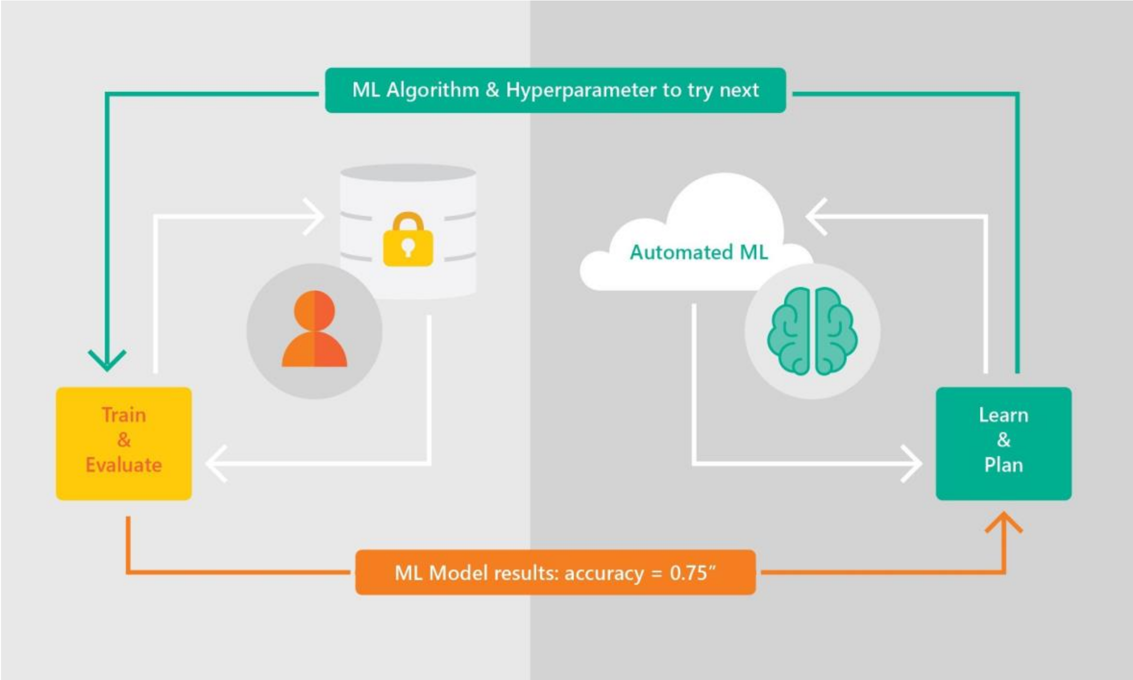
Benefits Overview

Azure Automated ML lets you

- Automate the exploration process
- Use resources more efficiently
- Optimize model for desired outcome
- Control resource budget

Apply it to different models and learning domains

- Pick training frameworks of choice
- Visualize all configurations in one place



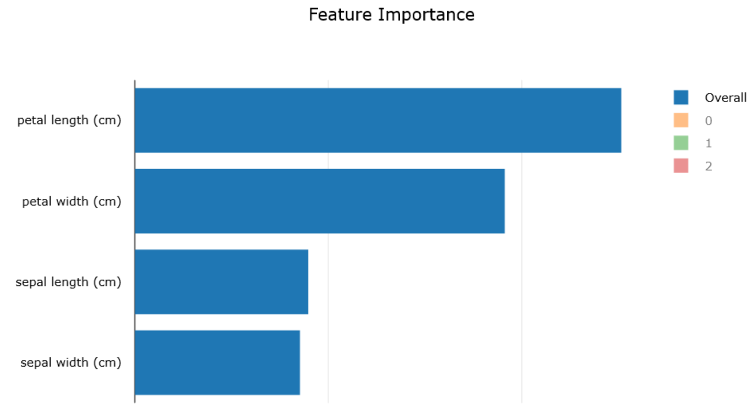
Note about security: on the right side of the automated ML service, the gray part is separated from the training and data, only the result (orange bottom block) is sent back from training to the service; hence your data and algorithm safely stay within your subscription.

Azure Automated ML

Model Explainability

```
automl_config = AutoMLConfig(task = 'classification',
                             debug_log = 'automl_errors.log',
                             primary_metric = 'AUC_weighted',
                             max_time_sec = 12000,
                             iterations = 10,
                             verbosity = logging.INFO,
                             X = X_train,
                             y = y_train,
                             X_valid = X_test,
                             y_valid = y_test,
                             model_explainability=True,
                             path=project_folder)
```

```
from azureml.train.automl.automlexplainer
import retrieve_model_explanation
shap_values, expected_values,
overall_summary, overall_imp,
per_class_summary, per_class_imp = \
retrieve_model_explanation(best_run)
#Overall feature importance
print(overall_imp) print(overall_summary)
#Class-level feature importance
print(per_class_imp)
print(per_class_summary)
```



You can view it in your workspace in Azure portal
Or you can show it using Jupyter widgets in a notebook:

```
from azureml.widgets import RunDetails
RunDetails(local_run).show()
```

Microsoft Research Paper & Examples

For those who wants to find out more about Automated Machine Learning:

https://arxiv.org/abs/1705.05355

https://github.com/Azure/MachineLearningNotebooks/tree/master/how-to-use-azureml/automated-machine-learning

Distributed Training with Azure ML Compute

Distributed Training with Azure ML Compute

You submit a model training ‘job’ – the infrastructure is managed for you.

- Jobs run on a VM or Docker container.
- Supports Low priority (Cheaper) or Dedicated (Reliable) VMS.

Auto-scales: Just specify min and max number of nodes.

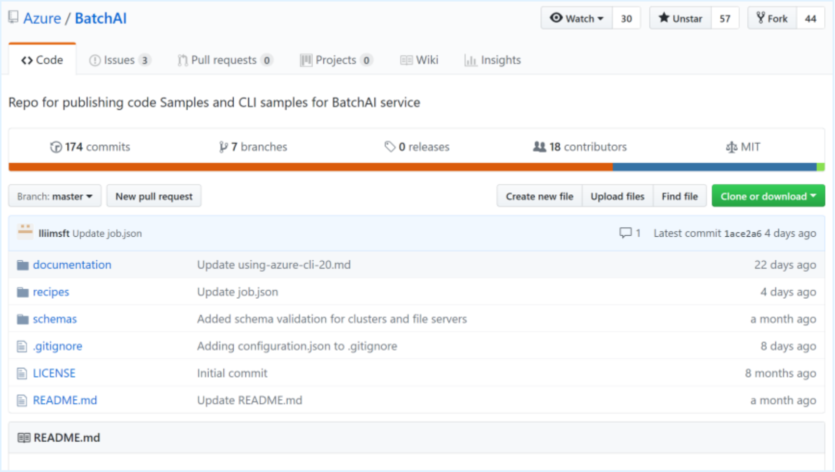
- If min is set to zero, *cluster is deleted when no jobs are running; so pay only for job duration.*

Works with most popular frameworks and multiple languages.

Supports [distributed training with Horovod](#).

Cluster can be shared; multiple experiments can be run in parallel.

Supports most VM Families, including latest NVidia GPUs for DL model training.



Try it for free!

http://aka.ms/amlfree

THANK YOU!

Learn more:
https://docs.microsoft.com/en-us/azure/machine-learning/service/

Visit the Getting started guide:
https://docs.microsoft.com/en-us/azure/machine-learning/service/quickstart-create-workspace-with-python

Fantastic free Azure notebooks (with Azure Machine Learning SDK pre-configured):
https://notebooks.azure.com