Azure Data Science Options

- Azure ML Studio -> Drag and drop. Understand for the exam. No code is needed there. Training and Deployment. Complete ML environment. Ideal for learning and beginner data scientists.
- Azure Databricks for Big Data - based on Spark. Massive scale with spark. User friendly portal. Dynamic scale. Secure collaboration (secured workspace). DS tools. You can use different languages in the same notebook.
    - Core artifacts: Jobs, libraries, clusters, workspaces and notebooks.
- Azure Data Science Virtual Machine - VM with almost all of the tools one would need to do DS already presintalled. You can deploy them directly to Azure and work from there. It's easy to customize for your needs. It has some sample code already there. They merged it with Deep Learning VM. There are specific versions for Geo data.
- SQL Server Machine Learning Services - We cna use this to analyze data on SQL Server. Useful for on-premise data. It is an option as source Python and R does not scale, security concerns, and operationalization.
- Spark on Azure HDInsight - massive scale with in-memory processing. Hortonworks Distribution. Easy management PaaS. Integration with other Azure services.

Databricks vs. HDInsights: Databricks it's easier to collaborate, built for collaboration and work in teams.

- Azure ML Service: core of the course. Model management, training, selection, hyper-param tuning, feature selection and model evaluation. It lets you automate tuning and selection tasks. All is in Python.

Azure Notebooks

Azure based Jupyter Notebooks. Free tier. Ready to use project that teach how to use Azure data and AI services.

Jupyter notebooks can be integrated in VScode and thus you can use git integration with that.

Azure notebooks only support Python, R and F#.

The advantage of Azure Notebooks is that most of the libraries are preinstalled, but you still have the possibility to install more libraries. You can upload data from your local machine and use custom environment configuration. By using VMs from your azure subscription you can add processing power.

It is Azure ML service ready. From Azure Notebooks you can call Azure ML Service.

Azure ML Service

Bring the power of containerization and automation to DS. Pack model and libraries into a container and run everything.

DS pipeline: Environment setup -> Data preparation -> Experimentation -> Deployment

- Environment setup: create a workspace to store your work. Use python or Azure portal. An experiment within the workspace stored model training information. Use the IDE of your choice.
- Data Preparation: Use python libraries or the Azure Data Prep SDK.
- Experimentation: Train models with Python open source modules of your choice. Train locally or in Azure. Submit model training to Azure containers. Monitor model training. Register final model.
- Deployment: to make a model available. Target deployment environments are: Docker images, Azure container instances, Azure kubernetes service, Azure IoT edge, Field Programmable Gate Array (FPGA). FOr the dpeloyment you'll need the following files:
    - A score script file tells Azure ML Services (AMLS) to call the model
    - An environment file specifies package dependencies
    - A configuration file requests the required resources for the container.

What is a Workspace

The top-level resource for AMLS. It serves as a hub for building and deploying models. You can create a workspace in the Azure portla, or you can create and access it using Python on an IDE of your choice.

All models must be registered in the workspace for future use. Together with the scoring scripts, you create an image for deployment.

The workspace stores experiment objects that are required for each model you create. Additionally, it saves your compute targets. You can track training runs.

What is an Image

An image has three components:

- A model and scoring script or application
- An environment file that declares the dependencies that are needed by the model, scoring script or application
- A configuration file that describes the necessary resources to execute the model

What is a Datastore

An abstraction over an Azure Storage account. Each workspace has a registered, default datastore that you can use right away, but you can register other Azure Blob or File storage containers as a datastore.

What is a Pipeline

A ML pipeline is a tool to create and manage workflows during a DS process: data manipulation, model training and testing and deployment phases. Each step of the process can run unattended in different compute targets, which makes it easier to allocate resources.

What is a Compute Target

Is the compute resource to run a training script or to host service deployment. It's attached to a workspace. Other than the local machine, users of the workspace share compute targets.

What is a deployed Web Service

For a deployed web service, you have the choices of container Instances, AKS or FPGAs. With your model, script and associated files all set in the image, you can create a web service.

OBS: Trainer said that its better to create a new RG for each ML workspace as there are several resources involved and we don't want to get a mess.

OBS2: In Azure Notebooks, change python kernel to 3.6, as the default is just set to Python 3! This can rise errors when importing azure ML libs.

Interact with ML Service

We can interact via Azure Notebook (linking the subscription), a visual interface, Notebook VMs and automated ML.

You can run notebooks in the workspace without any kind of authentication and they are stored in the WS. So it is useful to work in teams.

If you use JupyterLab, you can link a repo in Azure DevOps.

If you register a model with the same name multiple times, it gets uploaded with greater version.

OBS: Scalability is enabled during training, but once the code is deployed it is flat. Also, it is only supported as an Azure App Service so you keep paying even if it is idle.

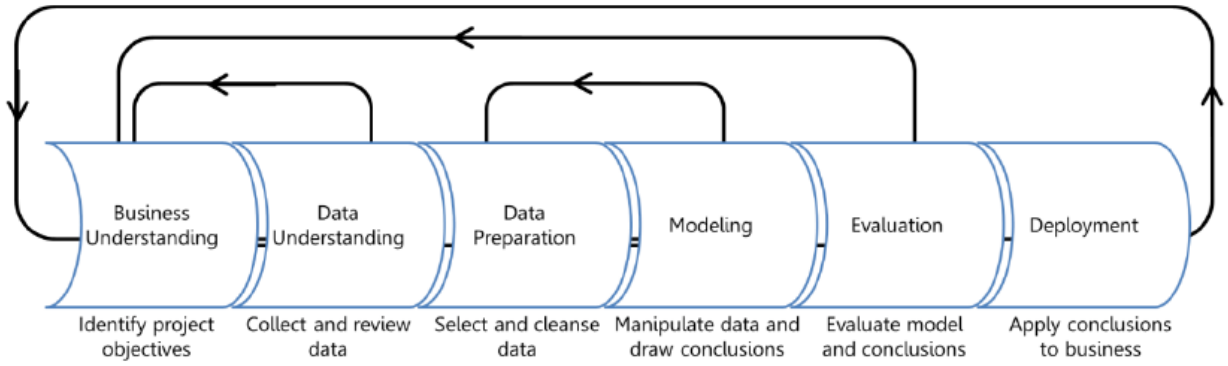
Explore AI solution development with data science services in Azure

Introduction to Data Science in Azure

Learning Objectives

- Learn the steps involved in the data science process
- Learn the machine learning modeling cycle
- Learn data cleansing and preparation
- Learn model feature engineering
- Learn model training and evaluation
- Learn about model deployment
- Discover the specialized roles in the data science process

The Data Science process



Iterative process that starts with a question, risen from business needs and understanding.

What is modeling?

Modeling is a cycle of data and business understanding. You start by exploring your assets, in this case data, with **Exploratory Data Analysis (EDA)**, from that point feature engineering starts and finally train a model on top, which is an algorithm that learns information and provides a probabilistic prediction.

In the end, the model is evaluated to check where it is accurate and where it is failing to correct the behavior.

Choose a use case

Identify the problem (business understanding) -> Define the project goals -> Identify data sources

Data preparation

Data cleansing and EDA are vital to the modeling process, to get insights on what data is or is not useful and what needs to be corrected or taken into account. Understanding the data is one of the most vital steps in the data science cycle.

Feature engineering

What extra knowledge we can extract by combining existing features to create new ones.

Model training

Split data -> Cross-validate data -> Obtain probabilistic prediction

Model evaluation

Hyperparameters are parameters used in model training that cannot be learned by the training process. These parameters must be set before model training begins.

For evaluating the results you need to set up a metric to compare different runs, such as accuracy or MSE.

Model deployment

Model deployment is the final stage of the data science procedure. It is often done by a developer, and is usually not part of the data scientist role.

Specialized roles in the Data Science process

In the data science process, there are specialists in each of the steps:

Business Analyst or Domain Expert, Data Engineer, Developer and Data Scientist.

Knowledge Check

1. Which of the following is not a specialized role in the Data Science Process?

- Database Administrator
- Data Scientist
- Data Engineer

Answer

DBA

1. Model feature engineering refers to which of the following?

- Selecting the best model to use for the experiment.

- Determine which data elements will help in making a prediction and preparing these columns to be used in model training.
- Exploring the data to understand it better.

  Answer

  Feature engineering involves the data scientist determining which data to use in model training and preparing the data so it can be used by the model.

1. The Model deployment involves.

- Calling a model to score new data.
- Training a model.
- Copying a trained model and its code dependencies to an environment where it will be used to score new data.

  Answer

  Deploying a model makes it available for use.

Choose the Data Science service in Azure you need

Show content

Learning Objectives

- Differentiate each of the Azure machine learning products.
- Identify key features of each product.
- Describe the use cases for each service.

Machine Learning options on Azure

We have the following services:

- **Azure Machine Learning Studio**: GUI-based solution best chosen for learning. It includes all DS pipeline steps, from importing and playing around with data to different deployment options. All is based in a drag-and-drop method.
- **Azure Databricks**: Great collaboration platform with a powerful notebook interface, job scheduling, AAD integration and granular security control. It allows to create and modify Spark clusters.
- **Azure Data Science Virtual Machine**: preconfigured VMs with lots of preinstalled popular ML tools. You can directly connect to the machine via ssh or remote desktop. There are different types of machines:

  - Linux and Windows OS, where Windows supports scalability with ML in SQL Server and Linux does not.
  - Deep Learning VM, offering DL tools.
  - Geo AI DSVM, with specific tools for working with spatial data. Includes ArcGIS.

- **SQL Server Machine Learning Services**: add-on which runs on the SQL Server on-premises and supports scale up and high performance of Python and R code. It includes several advantages:

  - Security, as the processing occurs closer to the data source.
  - Performance
  - Consistency
  - Efficiency, as you can use integrated tools such as PowerBI to report and analyze results.

- **Spark on HDInsight**: HDInsight is PaaS service offering Apache Hadoop. It provides several benefits:

  - Easy and fast to create and modify clusters on demand.
  - Usage of ETL tools in the cluster with Map Reduce and Spark.
  - Compliance standards with Azure Virtual Network, envryption and integration of Azure AD.
  - Integrated with other Azure services, such as ADLS or ADF.

  HDInsight Spark is an implementation of Apache Spark on Azure HDInsight.

- **Azure Machine Learning Service**: Supports the whole DS pipeline integration, scale processing and automate the following tasks:

  - Model management
  - Model training
  - Model selection
  - Hyper-parameter tuning
  - Feature selection
  - Model evaluation

  It supports open-source technologies such as Python and common ds tools. It makes it easier to containerize and deploy the model and automate several tasks. The platform is designed to support three roles:

  - Data Engineer to ingest and prepare data for analysis either locally or on Azure containers.

  - Data Scientist to apply the modeling tools and processes. AMLS support sklearn, tensorFlow, pyTorch, Microsoft Cognitive Toolkit and Apache MXNet.

  - Developer to create an image of the built and trained model with all the needed components. An **image** contains:

    a. The model
    b. A scoring script or application which passes input to the model and returns the output of the model

c. The required dependencies, such as Python scripts or packages needed by the model or scoring script.

Images can be deployed as Docker images or field programmable gate array (FPGA) images. Iages can be deployed to a web service (running in Azure Container Instance, FPGA or Azure Kubernetes Service), or an IoT module (IoT Edge).

OBS: Scalability is enabled during training, but once the code is deployed it is flat. Also, it is only supported as an Azure App Service so you keep paying even if it is idle.

Knowledge Check

1. Azure Machine Learning service supports which programming language.

- R
- Julia
- Python

Answer

Python is supported by Azure Machine Learning service.

1. Azure Databricks is built on which Big Data platform?

- Azure SQL Data Warehouse
- SQL Server
- Apache Spark

Answer

Azure Databricks makes using Spark easier.

1. Which is not an operating system available for an Azure Data Science Virtual Machine?

- Windows
- Linux
- Apple iOS

Answer

Data Science VMs running Apple iOS are not available.

Build AI solutions with Azure Machine Learning service

Introduction to Azure Machine Learning service

Show content

Learning Objectives

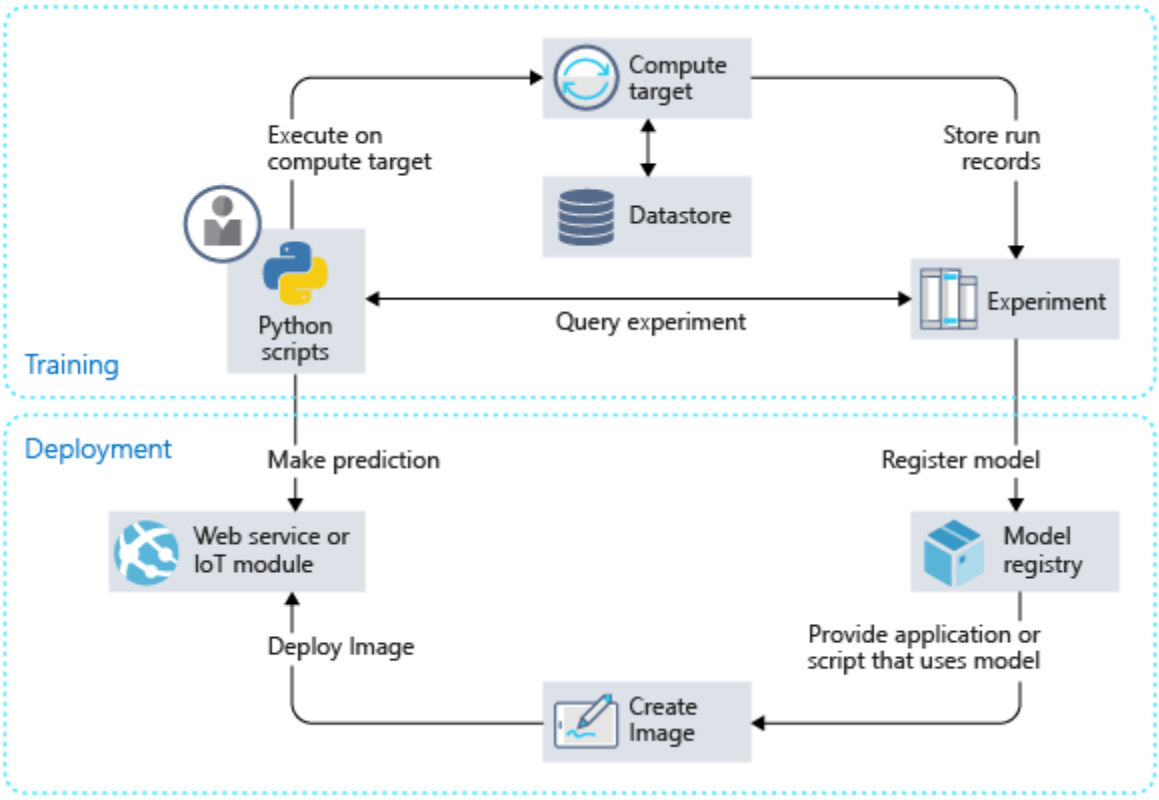
- Learn the difference between Azure Machine Learning Studio and Azure Machine Learning service
- See how Azure Machine Learning service fits into the data science process
- Learn the concepts related to an Azure Machine Learning service experiment
- Explore the Azure Machine Learning service pipeline
- Train a model using Azure Machine Learning service

Azure Machine Learning Service within a data science process

Environment Set Up -> Data Preparation -> Experimentation -> Deployment

- **Environment setup**: First step is creating a **Workspace**, where you store your ML work. An **Experiment** is created within the workspace to store information about runs for your model. You can have multiple experiments in one workspace. You can interact with the environment with different IDEs such as PyCharm or Azure Notebooks.
- **Data Preparation**: explore, analyze and visualize the sources. You can use any tool. Azure provides the following SDK Azureml.dataprep.
- **Experimentation**: Iterative process of training and testing. With AMLS you can run the model in Azure containers. You need to create and configure a computer target object used to provision computer resources.
- **Deployment**: Create a Docker image that will get deployed to Azure Container Instances (you could also choose AKS, Azure IoT or FPGA).

Create a machine learning experiment



- **Workspace**: top-level resource in AMLS where you build and deploy your models. With a registered model and scoring scripts you can create an image for deployment. It stores experiment objects which save computer targets, track runs, logs, metrics and outputs.
- **Image**: it has three key components:
  i. A model and scoring script or application
  ii. An environment file that declares the dependencies.
  iii. A configuration file with the necessary resources to execute the model.
- **Datastore**: Abstraction over an Azure Storage account. Each workspace has a default one, but you could add Blob or File storage containers.
- **Pipeline**: Tool to create and manage workflows during a ds process. Each step can run unattended in different computer targets, which makes it easier to allocate resources.
- **Computer target**: Resource to run a training model or to host service deployment. It is attached to a workspace.
- **Deployed Web service**: You can choose between ACI, AKS or FPGA. With the model, script and image files you can create a Web service.
- **IoT module**: It is a Docker container and has the same needs as a Web Service. It enables to monitor a hosting device.

Creating a pipeline

Some features or Azure ML pipelines are:

- Schedule tasks and executions,
- You can allocate different computer targets for different steps and coordinate multiple pipelines,

- You can reuse pipeline scripts and customize them,
- You can record and manage input, output, intermediate tasks and data.

Knowledge Check

1. The Azure Machine Learning service SDK is which of the following?

- A visual machine learning development portal.
- A Python package containing functions to use the Azure ML service.
- A special type of Azure virtual machine.

Answer

The modules provided by the Azure ML SDK provide the functions you need to work with the service in Python.

1. Which of the following is the underlying technology of the Azure Machine Learning service?

- Spark
- Hadoop
- Containerization including Docker and Kubernetes

Answer

Containerization is a key technology used by the Azure ML service.

1. Which of the following is not a component of an Azure Machine Learning service workspace image?

- An R package
- An environment file that declares dependencies that are needed by the model, scoring script or application.
- A model scoring script

Answer

R packages are not part of an Azure Machine Learning service workspace image.

1. Which of the following descriptions accurately describes Azure Machine Learning?

    - A Python library that you can use as an alternative to common machine learning frameworks like Scikit-Learn, PyTorch, and Tensorflow.
    - A cloud-based platform for operating machine learning solutions at scale.

- An application for Microsoft Windows that enables you to create machine learning models by using a drag and drop interface.

Answer

Cloud based Platform: Azure Machine Learning enables you to manage machine learning model data preparation, training, validation, and deployment. It supports existing frameworks such as Scikit-Learn, PyTorch, and Tensorflow; and provides a cross-platform platform for operationalizing machine learning in the cloud.

2. Which edition of Azure Machine Learning workspace should you provision if you only plan to use the graphical Designer tool to train machine learning models?

   - Basic
   - Enterprise

   Answer

   The visual Designer tool is not available in Basic edition workspaces, so you must create an Enterprise workspace to use it.

3. You are using the Azure Machine Learning Python SDK to write code for an experiment. You must log metrics from each run of the experiment, and be able to retrieve them easily from each run. What should you do?

   - Add print statements to the experiment code to print the metrics.
   - Save the experiment data in the outputs folder.
   - Use the log* methods of the Run class to record named metrics.

   Answer

   To record metrics in an experiment run, use the Run.log* methods.

Train a local ML model with Azure Machine Learning service

Show content

Learning Objectives

- Use an Estimator to run a model training script as an Azure Machine Learning experiment.
- Create reusable, parameterized training scripts.
- Register models, including metadata such as performance metrics.

As this is a rather practical module, you can refer to the labs notebooks or directly to Azure's docs.

What is HyperDrive

HyperDrive is a built-in service that automatically launches multiple experiments in parallel each with different parameter configurations. Azure Machine Learning then automatically finds the configuration that results in the best performance measured by the metric you choose. The service will terminate poorly performing training runs to minimize compute resources usage.

Azure Machine Learning estimators

In Azure Machine Learning, you can use a **Run Configuration** and a **Script Run Configuration** to run a script-based experiment that trains a machine learning model. However, these configurations may end up being really complex, so another abstraction layer is added: An **Estimator** encapsulates a run configuration and a script configuration in a single object.

We have some default Estimators for frameworks such as Scikit Learn, Pytorch and TF.

Writing a Script to Train a Model

After training a model, it should be saved in the **outputs** directory. For example witch SKlearn:

```
from azureml.core import Run
import joblib

# Get the experiment run context
run = Run.get_context()

# Train and test...

# Save the trained model
os.makedirs('outputs', exist_ok=True)
joblib.dump(value=model, filename='outputs/model.pkl')

run.complete()
```

Using an Estimator

You can use a generic Estimator class to define a run configuration for a training script like this:

```
from azureml.train.estimator import Estimator
from azureml.core import Experiment

# Create an estimator
estimator = Estimator(source_directory='experiment_folder',
                      entry_script='training_script.py',
                      compute_target='local',
                      conda_packages=['scikit-learn']
                      )
```

```
# Or use a framework specific estimator as
estimator = SKLearn(source_directory='experiment_folder',
                    entry_script='training_script.py'
                    compute_target='local'
                    )

# Create and run an experiment
experiment = Experiment(workspace = ws, name = 'training_experiment')
run = experiment.submit(config=estimator)
```

Using script parameters

Used to increase the flexibility of script-based experiments.

These parameters are read as usual Python parameters in scripts. So for example, after setting the Run:

```
# Set regularization hyperparameter
parser = argparse.ArgumentParser()
parser.add_argument('--reg_rate', type=float, dest='reg', default=0.01)
args = parser.parse_args()
reg = args.reg
```

To use parameters in **Estimators**, add the script_params value as a dict:

```
# Create an estimator
estimator = SKLearn(source_directory='experiment_folder',
                    entry_script='training_script.py',
                    script_params = {'--reg_rate': 0.1},
                    compute_target='local'
                    )
```

Registering models

After running an experiment that trains a model you can use a reference to the Run object to retrieve its outputs, including the trained model.

Retrieving Model Files

From the run object we can get all the files that it generated with run.get_file_names() and download the models as (recall how we said that usually those were stored under outputs/)

```
run.download_file(name='outputs/model.pkl', output_file_path='model.pkl')
```

Registering a Model

With Model.register() we can save different versions of our models:

```
from azureml.core import Model

model = Model.register(workspace=ws,
                       model_name='classification_model',
                       model_path='model.pkl', # local path
                       description='A classification model',
                       tags={'dept': 'sales'},
                       model_framework=Model.Framework.SCIKITLEARN,
```

```
                model_framework_version='0.20.3')
```

Or the same by referencing the run object:

```
run.register_model( model_name='classification_model',
                    model_path='outputs/model.pkl', # run outputs path
                    description='A classification model',
                    tags={'dept': 'sales'},
                    model_framework=Model.Framework.SCIKITLEARN,
                    model_framework_version='0.20.3')
```

We can then view all the models we saved by using:

```
for model in Model.list(ws):
  # Get model name and auto-generated version
  print(model.name, 'version:', model.version)
```

Knowledge Check

1. An Experiment contains which of the following?

    - A composition of a series of runs
    - A Docker image
    - The data used for model training

    Answer

    A composition of a series of runs: Azure ML Studio provides a visual drag and drop machine learning development portal but that is a separate offering.

2. A run refers to which of the following?

    - Python code for a specific task such as training a model or tuning hyperparameters. Run does the job of logging metrics and uploading the results to Azure platform.
    - A set of containers managed by Kubertes to run your models.
    - A Spark cluster.

    Answer

    Python code for a specific task such as training a model or tuning hyperparameters. Run does the job of logging metrics and uploading the results to Azure platform.

3. A hyperparameter is which of the following?

    - A model parameter that cannot be learned by the model training process.
    - A model feature derived from the source data.
    - A parameter that automatically and frequently changes value during a single model training run.

    Answer

Hyperparameters control how the model training executes and must be set before model training.

4. Before you can train and run experiments in your code, you must do which of the following?

    - Create a virtual machine
    - Log out of the Azure portal
    - Write a model scoring script

    Answer

    Your Python script needs to connect to the Azure ML workspace before you can train and run experiments.

5. Which of the following is a technique for determining hyperparameter values?

    - grid searching
    - Bayesian sampling
    - hyper searching

    Answer

    Grid searching is often used by data scientists to find the best hyperparamter value.

6. You have written a script that uses the Scikit-Learn framework to train a model. Which framework-specific estimator should you use to run the script as an experiment?

    - PyTorch
    - Tensorflow
    - SKLearn

    Answer

    To run a scikit-learn training script as an experiment, use the generic Estimator estimator or a SKLearn estimator.

7. You have run an experiment to train a model. You want the model to be stored in the workspace, and available to other experiments and published services. What should you do?

    - Register the model in the workspace.
    - Save the model as a file in a Compute Instance.
    - Save the experiment script as a notebook.

    Answer

To store a model in the workspace, register it.

Working with Data in Azure Machine Learning

Show content

Learning objectives

- Create and use datastores
- Create and use datasets

Introduction to datastores

Abstractions for cloud data sources. They hold the connection information and can be used to both read and write. The different sources could be (sample from here):

- Azure Storage (blob and file containers)
- Azure Data Lake Storage
- Azure SQL Database
- Azure Databricks file system (DBFS)

Using datastores

Each workspace has two built-in datastores (blob container + Azure Storage File container) used as system storage by AMLS. You have a limited use on top of those.

The good part of using external datasources - which is the usual - is the ability to share data accross multiple experiments, regardless of the compute context in which those experiments are running.

You can use the AMLS SDK to store / retrieve data from the datastores.

Registering a datastore

To register a datastore, you could either use the UI in AMLS or the SDK:

```
from azureml.core import Workspace, Datastore

ws = Workspace.from_config()

# Register a new datastore
blob_ds = Datastore.register_azure_blob_container(
   workspace=ws,
   datastore_name='blob_data',
   container_name='data_container',
```

```
    account_name='az_store_acct',
    account_key='123456abcde789…'
)
```

Managing datastores

Again, managing can be done via UI or SDK:

```
# list
for ds_name in ws.datastores:
    print(ds_name)

# get
blob_store = Datastore.get(ws, datastore_name='blob_data')

# get default
default_store = ws.get_default_datastore()

# set default
ws.set_default_datastore('blob_data')
```

Use datastores

You can interact directly with a datastore via the SDK and *pass data references* to scripts that need to access data.

OBS: For blobs to work correctly as a datastore and be accessible in the code to upload / download, the storage account should be Standard / Hot, not Premium!

Working directly with a datastore

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

blob_ds.download(target_path='downloads',
                 prefix='/data',
                 show_progress=True)
```

Using data references

When you want to use a datastore in an experiment script, you must pass a data reference to the script. There are the following accesses:

- **Download**: Contents are downloaded to the compute context.
- **Upload**: The files generated by the experiment are uploaded to the datastore after the run completes.
- **Mount**: When experiments run on a remote compute (not local), you can mount the path.

To pass the reference to an experiment script, define the script_params:

```
data_ref = blob_ds.path('data/files').as_download(path_on_compute='training_data')
```

```
estimator = SKLearn(source_directory='experiment_folder',
                    entry_script='training_script.py'
                    compute_target='local',
                    script_params = {'--data_folder': data_ref})
```

script_params can then be retrieved via argparse.

Introduction to datasets

Datasets are versioned packaged data objects that can be easily consumed in experiments and pipelines. They are the recommended way to work with data.

Datasets can be based on files in a datastore or on URLs and other resources.

Types of dataset

- **Tabular**: useful when when we work, for example, with pandas.
- **File**: For unstructured data. Dataset will present a list of paths that can be read as thought from the file system. For example, for images in a CNN.

Creating and registering datasets

You can use the UI or the SDK to create datasets from files or paths (which can include wildcards * for regex).

Creating and registering tabular datasets

```
from azureml.core import Dataset

blob_ds = ws.get_default_datastore()
csv_paths = [(blob_ds, 'data/files/current_data.csv'),
             (blob_ds, 'data/files/archive/*.csv')]
tab_ds = Dataset.Tabular.from_delimited_files(path=csv_paths)
tab_ds = tab_ds.register(workspace=ws, name='csv_table')
```

Creating and registering file datasets

```
from azureml.core import Dataset

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')
```

Retrieving a registered dataset

You can retrieve datasets by the datasets attribute of a Workspace or by calling get_by_name or get_by_id of the Dataset class:

```
import azureml.core
from azureml.core import Workspace, Dataset

# Load the workspace from the saved config file
```

```
ws = Workspace.from_config()

# Get a dataset from the workspace datasets collection
ds1 = ws.datasets['csv_table']

# Get a dataset by name from the datasets class
ds2 = Dataset.get_by_name(ws, 'img_files')
```

Dataset versioning

Useful to reproduce experiments with data in the same state. Use the create_new_version property when registering a dataset:

```
img_paths = [(blob_ds, 'data/files/images/*.jpg'),
         (blob_ds, 'data/files/images/*.png')]
file_ds = Dataset.File.from_files(path=img_paths)
file_ds = file_ds.register(workspace=ws, name='img_files', create_new_version=True)
```

To retrieve a specific version:

```
img_ds = Dataset.get_by_name(workspace=ws, name='img_files', version=2)
```

Use datasets

You can read data directly from a dataset, or you can pass a dataset as a named input to a script configuration or estimator.

Working with a dataset directly

If you have a reference to a dataset, you can access its contents directly.

```
df = tab_ds.to_pandas_dataframe()
```

When working with a file dataset, use to_path():

```
for file_path in file_ds.to_path():
   print(file_path)
```

Passing a dataset to an experiment script

When you need to access a dataset in an experiment script, you can pass the dataset as an input to a **ScriptRunConfig** or an **Estimator**:

```
estimator = SKLearn( source_directory='experiment_folder',
                entry_script='training_script.py',
                compute_target='local',
                inputs=[tab_ds.as_named_input('csv_data')],
                pip_packages=['azureml-dataprep[pandas]')
```

Since the script will need to work with a **Dataset** object, you must include either the full **azureml-sdk** package or the **azureml-dataprep** package with the **pandas** extra library in the script's compute environment.

Then, in the experiment

```
run = Run.get_context()
```

```
data = run.input_datasets['csv_data'].to_pandas_dataframe()
```

Finally, when passing a file dataset, you must specify the access mode:

```
estimator = Estimator( source_directory='experiment_folder',
                entry_script='training_script.py'
                compute_target='local',
                inputs=[img_ds.as_named_input('img_data').as_download(path_on_compute='data')],
                pip_packages=['azureml-dataprep[pandas]')
```

Knowledge Check

1. You've uploaded some data files to a folder in a blob container, and registered the blob container as a datastore in your Azure Machine Learning workspace. You want to run a script as an experiment that loads the data files and trains a model. What should you do?

    - Save the experiment script in the same blob folder as the data files.
    - Create a data reference for the datastore location and pass it to the script as a parameter.
    - Create global variables for the Azure Storage account name and key in the experiment script.

    Answer

    To access a path in a datastore in an experiment script, you must create a data reference and pass it to the script as a parameter. The script can then read data from the data reference parameter just like a local file path.

2. You've registered a dataset in your workspace. You want to use the dataset in an experiment script that is run using an estimator. What should you do?

    - Pass the dataset as a named input to the estimator.
    - Create a data reference for the datastore location where the dataset data is stored, and pass it to the script as a parameter.
    - Use the dataset to save the data as a CSV file in the experiment script folder before running the experiment.

    Answer

    To access a dataset in an experiment script, pass the dataset as a named input to the estimator.

Working with Compute Contexts in Azure Machine Learning

Show content

Learning objectives

- Create and use environments.
- Create and use compute targets.

Introduction to environments

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

Environments in Azure Machine Learning

In general, AML handles environment creationm, package installation and environment registration for you - usually through the creation of Docker containers. You'd just need to specify the packages you want. You could also manage the environments if needed.

Environments are encapsulated by the **Environment** class; which you can use to create environments and specify runtime configuration for an experiment.

Creating environments

- **Creating an environment from a specification file**: based on conda or pip. For example, a file named **conda.yml**

```
name: py_env
   dependencies:
   - numpy
   - pandas
   - scikit-learn
   - pip:
      - azureml-defaults
```

Then, create the environment with the SDK

```
from azureml.core import Environment

env = Environment.from_conda_specification(name='training_environment',
                                           file_path='./conda.yml')
```

- **Creating an environment from an existing Conda environment**: If you have already a defined Conda environment on the workstation you can reuse it in AML

```
from azureml.core import Environment

env = Environment.from_existing_conda_environment(name='training_environment',
                                                  conda_environment_name='py_env')
```

- **Creating an environment by specifying packages**: using a **CondaDependencies** object:

```
from azureml.core import Environment
from azureml.core.conda_dependencies import CondaDependencies

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

Registering and reusing environments

After you've created an environment, you can register it in your workspace and reuse it for future experiments that have the same Python dependencies.

Register it via `env.register(workspace=ws)` and get the registered environments in a workspace using `Environment.list(workspace=ws)`.

Retrieving and using an environment

You can retrieve an environment and assign it to an **Estimator** or a **ScriptRunConfig**:

```
from azureml.core import Environment, Estimator

training_env = Environment.get(workspace=ws, name='training_environment')
estimator = Estimator(source_directory='experiment_folder'
                      entry_script='training_script.py',
                      compute_target='local',
                      environment_definition=training_env)
```

OBS: When an experiment based on the estimator is run, Azure Machine Learning will look for an existing environment that matches the definition, and if none is found a new environment will be created based on the registered environment specification.

Introduction to compute targets

Compute Targets are physical or virtual computers on which experiments are run. You can assign experiments to specific compute targets. This means that one can test on cheaper ones and run individual processes on GPUs, if needed.

You pay-by-use as compute targets

- Start on-demand and stop automatically when no longer required.
- Scale automatically based on workload processing needs (for model training)

Types of compute

- **Local compute**: Great for test and development. The experiment will run where the code is initiated, e.g., you own computer or a VM with jupyter on top.
- **Training Clusters**: multi-node clusters of VMs that automatically scale up or down to meet demand for training workloads. Useful when working with large data or when needing parallel processing.

- **Inference clusters**: To deploy trained models as production services. They use containerization to enable rapid initialization of compute for on-demand inferencing.
- **Attached compute**: You can attach another Azure-based compute environment to AML, as another VM or a Databricks cluster. They can be used for certain types of workload.

More info here.

Create compute targets

Can be done via UI or SDK. UI is the most common.

Creating a managed compute target with the SDK

They are managed by AML, e.g., a training cluster.

```
from azureml.core import Workspace
from azureml.core.compute import ComputeTarget, AmlCompute

# Load the workspace from the saved config file
ws = Workspace.from_config()

# Specify a name for the compute (unique within the workspace)
compute_name = 'aml-cluster'

# Define compute configuration
compute_config = AmlCompute.provisioning_configuration(vm_size='STANDARD_DS12_V2',
                                                       min_nodes=0, max_nodes=4,
                                                       vm_priority='dedicated')

# Create the compute
aml_cluster = ComputeTarget.create(ws, compute_name, compute_config)
aml_cluster.wait_for_completion(show_output=True)
```

Priority can be **dedicated** to use for this cluster or **low priority**, for less cost but the possibility to be preemted.

Attaching an unmanaged compute target with the SDK

Unmanaged instances are defined and managed outside of the AML, e.g., a VM or a Databricks.

```
from azureml.core import Workspace
from azureml.core.compute import ComputeTarget, DatabricksCompute

# Load the workspace from the saved config file
ws = Workspace.from_config()

# Specify a name for the compute (unique within the workspace)
compute_name = 'db_cluster'

# Define configuration for existing Azure Databricks cluster
db_workspace_name = 'db_workspace'
db_resource_group = 'db_resource_group'
```

```
db_access_token = '1234-abc-5678-defg-90...'
db_config = DatabricksCompute.attach_configuration(resource_group=db_resource_group,
                                    workspace_name=db_workspace_name,
                                    access_token=db_access_token)

# Create the compute
databricks_compute = ComputeTarget.attach(ws, compute_name, db_config)
databricks_compute.wait_for_completion(True)
```

Checking for an existing compute target

You can check if a compute targets exists to only create it otherwise:

```
from azureml.core.compute import ComputeTarget, AmlCompute
from azureml.core.compute_target import ComputeTargetException

compute_name = "aml-cluster"

# Check if the compute target exists
try:
   aml_cluster = ComputeTarget(workspace=ws, name=compute_name)
   print('Found existing cluster.')
except ComputeTargetException:
   # If not, create it
   compute_config = AmlCompute.provisioning_configuration(vm_size='STANDARD_DS12_V2',
                                       max_nodes=4)
   aml_cluster = ComputeTarget.create(ws, compute_name, compute_config)

aml_cluster.wait_for_completion(show_output=True)
```

More info here.

Use compute targets

You can use them to run specific workloads:

```
from azureml.core import Environment, Estimator

compute_name = 'aml-cluster'

training_env = Environment.get(workspace=ws, name='training_environment')

estimator = Estimator(source_directory='experiment_folder',
                entry_script='training_script.py',
                environment_definition=training_env,
                compute_target=compute_name)
```

OBS: When an experiment for the estimator is submitted, the run will be queued while the compute target is started and the specified environment deployed to it, and then the run will be processed on the compute environment.
Instead of working by name, you could also pass a **ComputeTarget** object:

```
from azureml.core import Environment, Estimator
from azureml.core.compute import ComputeTarget

compute_name = 'aml-cluster'
```

```
training_cluster = ComputeTarget(workspace=ws, name=compute_name)

training_env = Environment.get(workspace=ws, name='training_environment')

estimator = Estimator(source_directory='experiment_folder',
                      entry_script='training_script.py',
                      environment_definition=training_env,
                      compute_target=training_cluster)
```

Knowledge Check

1. You're using the Azure Machine Learning Python SDK to run experiments. You need to create an environment from a Conda configuration (.yml) file. Which method of the Environment class should you use?

    - create
    - create_from_conda_specification
    - create_from_existing_conda_environment

    Answer

    Use the create_from_conda_specification method to create an environment from a configuration file. The create method requires you to explicitly specify conda and pip packages, and the create_from_existing_conda_environment requires an existing environment on the computer.

2. You must create a compute target for training experiments that require a graphical processing unit (GPU). You want to be able to scale the compute so that multiple nodes are started automatically as required. Which kind of compute target should you create?

    - Compute Instance
    - Training Cluster
    - Inference Cluster

    Answer

    Use a training cluster to create multiple nodes of GPU-enabled VMs that are started automatically as needed.

Orchestrating machine learning with pipelines

Show content

Learning objectives

- Create an Azure Machine Learning pipeline.
- Publish an Azure Machine Learning pipeline.
- Schedule an Azure Machine Learning pipeline.

Introduction to pipelines

A pipeline is a workflow of machine learning tasks in which each task is implemented as a step. Steps can be sequential or parallel and you can choose a specific compute target for them to run on.

A pipeline can be executed as a process by running the pipeline as an experiment.

They can be triggered via an scheduler or through a REST endpoint.

Pipeline steps

There are different types of steps:

- **PythonScriptStep**: runs a specific python script.
- **EstimatorStep**: runs an estimator.
- **DataTransferStep**: Uses Azure Data Factory to copy data between data stores.
- **DatabricksStep**: runs a notebook, script or compiled JAR on dbks.
- **AdlaStep**: runs a U-SQL job in Azure Data Lake Analytics.

You can find the full list here.

Defining steps in a pipeline

First, you define the steps and then assemble the pipeline based on those:

```
from azureml.pipeline.steps import PythonScriptStep, EstimatorStep

# Step to run a Python script
step1 = PythonScriptStep(name = 'prepare data',
                         source_directory = 'scripts',
                         script_name = 'data_prep.py',
                         compute_target = 'aml-cluster',
                         runconfig = run_config)

# Step to run an estimator
step2 = EstimatorStep(name = 'train model',
                      estimator = sk_estimator,
                      compute_target = 'aml-cluster')

from azureml.pipeline.core import Pipeline
from azureml.core import Experiment

# Construct the pipeline
train_pipeline = Pipeline(workspace = ws, steps = [step1,step2])
```

```
# Create an experiment and run the pipeline
experiment = Experiment(workspace = ws, name = 'training-pipeline')
pipeline_run = experiment.submit(train_pipeline)
```

Pass data between pipeline steps

It is not unusual to have steps depending on previous steps' results.

The PipelineData object

The **PipelineData** object is a special kind of **DataReference** that:

- References a location in a datastore.
- Creates a data dependency between pipeline steps.

It is an intermediary store between two subsequent steps: step1 -> PipelineData -> step2.

PipelineData step inputs and outputs

To use a **PipelineData** object you must:

1. Define a named **PipelineData** object that references a location in a datastore.
2. Configure the input / output of the steps that use it.
3. Pass the **PipelineData** object as a script parameter in steps that run scripts (and add the argparse in those scripts, as we do with usual data refs).

```
from azureml.pipeline.core import PipelineData
from azureml.pipeline.steps import PythonScriptStep, EstimatorStep

# Get a dataset for the initial data
raw_ds = Dataset.get_by_name(ws, 'raw_dataset')

# Define a PipelineData object to pass data between steps
data_store = ws.get_default_datastore()
prepped_data = PipelineData('prepped',  datastore=data_store)

# Step to run a Python script
step1 = PythonScriptStep(name = 'prepare data',
                         source_directory = 'scripts',
                         script_name = 'data_prep.py',
                         compute_target = 'aml-cluster',
                         runconfig = run_config,
                         # Specify dataset as initial input
                         inputs=[raw_ds.as_named_input('raw_data')],
                         # Specify PipelineData as output
                         outputs=[prepped_data],
                         # Also pass as data reference to script
                         arguments = ['--folder', prepped_data])

# Step to run an estimator
step2 = EstimatorStep(name = 'train model',
                      estimator = sk_estimator,
```

```
    compute_target = 'aml-cluster',
    # Specify PipelineData as input
    inputs=[prepped_data],
    # Pass as data reference to estimator script
    estimator_entry_script_arguments=['--folder', prepped_data])
```

Reuse pipeline steps

AML includes some caching and reuse feature to reduce the time to run some steps.

Managing step output reuse

By default, the step output from a previous pipeline run is reused without rerunning the step. This is useful if the scripts, sources and directories have no change at all, otherwise this may lead to stale results.

To control reuse for an individual step, you can use allow_reuse parameter:

```
step1 = PythonScriptStep(name = 'prepare data',
                         ...
                         # Disable step reuse
                         allow_reuse = False)
```

Forcing all steps to run

You can force all steps to run regardless of individual reuse by setting the regenerate_outputs param at submision time:

```
pipeline_run = experiment.submit(train_pipeline, regenerate_outputs=True)
```

Publish pipelines

After you have created a pipeline, you can publish it to create a REST endpoint through which the pipeline can be run on demand.

```
published_pipeline = pipeline.publish(name='training_pipeline',
                                      description='Model training pipeline',
                                      version='1.0')
```

You can also publish the pipeline on a successful run:

```
# Get the most recent run of the pipeline
pipeline_experiment = ws.experiments.get('training-pipeline')
run = list(pipeline_experiment.get_runs())[0]

# Publish the pipeline from the run
published_pipeline = run.publish_pipeline(name='training_pipeline',
                                          description='Model training pipeline',
                                          version='1.0')
```

To get the endpoint

```
rest_endpoint = published_pipeline.endpoint
print(rest_endpoint)
```

Using a published pipeline

To use the endpoint, you need to get the token from a service principal with permission to run the pipeline.

```
import requests

response = requests.post(rest_endpoint,
                         headers=auth_header,
                         json={"ExperimentName": "run_training_pipeline"})
run_id = response.json()["Id"]
print(run_id)
```

Use pipeline parameters

To define parameters for a pipeline, create a **PipelineParameter** object for each parameter, and specify each parameter in at least one step.

```
from azureml.pipeline.core.graph import PipelineParameter

reg_param = PipelineParameter(name='reg_rate', default_value=0.01)

...

step2 = EstimatorStep(name = 'train model',
                      estimator = sk_estimator,
                      compute_target = 'aml-cluster',
                      inputs=[prepped],
                      estimator_entry_script_arguments=['--folder', prepped,
                                                        '--reg', reg_param])
```

OBS: You must define parameters for a pipeline before publishing it.

Running a pipeline with a parameter

After publishing a pipeline with a parameter, you can specify it in the JSON payload in the REST call:

```
response = requests.post(rest_endpoint,
                         headers=auth_header,
                         json={"ExperimentName": "run_training_pipeline",
                             "ParameterAssignments": {"reg_rate": 0.1}})
```

Schedule pipelines

Scheduling a pipeline for periodic intervals

To schedule a pipeline to run at periodic intervals, you must define a **ScheduleRecurrance** that determines the run frequency, and use it to create a **Schedule**.

```
from azureml.pipeline.core import ScheduleRecurrence, Schedule
```

```
daily = ScheduleRecurrence(frequency='Day', interval=1)
pipeline_schedule = Schedule.create(ws, name='Daily Training',
                                    description='trains model every day',
                                    pipeline_id=published_pipeline.id,
                                    experiment_name='Training_Pipeline',
                                    # daily schedule
                                    recurrence=daily)
```

Triggering a pipeline run on data changes

You can also monitor a specified path on a datastore. This will become a trigger for a new run.

```
from azureml.core import Datastore
from azureml.pipeline.core import Schedule

training_datastore = Datastore(workspace=ws, name='blob_data')
pipeline_schedule = Schedule.create(ws, name='Reactive Training',
                                    description='trains model on data change',
                                    pipeline_id=published_pipeline_id,
                                    experiment_name='Training_Pipeline',
                                    datastore=training_datastore,
                                    path_on_datastore='data/training')
```

Knowledge Check

1. You're creating a pipeline that includes two steps. Step 1 preprocesses some data, and step 2 uses the preprocessed data to train a model. What type of object should you use to pass data from step 1 to step 2 and create a dependency between these steps?

    - Datastore
    - PipelineData
    - Data Reference

    Answer

    To pass data between steps in a pipeline, use a PipelineData object.

2. You've published a pipeline that you want to run every week. You plan to use the Schedule.create method to create the schedule. What kind of object must you create first to configure how frequently the pipeline runs?

    - Datastore
    - PipelineParameter
    - ScheduleRecurrance

    Answer

You need a ScheduleRecurrance object to create a schedule that runs at a regular interval.

Deploying machine learning models with Azure Machine Learning

Show content

Learning objectives

- Deploy a model as a real-time inferencing service.
- Consume a real-time inferencing service.
- Troubleshoot service deployment

Deploying a model as a real-time service

You can deploy a model as a real-time web service to several kinds of compute target:

- Local compute
- Azure ML compute instance
- Azure Container Instance (ACI)
- AKS
- Azure Function
- IoT module

AML uses containers for model packaging and deployment.

1. Register a trained model

After a successful training, you first need to register the model.

To register from a local file:

```
from azureml.core import Model

classification_model = Model.register(workspace=ws,
                       model_name='classification_model',
                       model_path='model.pkl', # local path
                       description='A classification model')
```

Or to reference to the **Run** used to train the model:

```
run.register_model( model_name='classification_model',
                    model_path='outputs/model.pkl', # run outputs path
                    description='A classification model')
```

2. Define an Inference Configuration

The model will be deployed as a service that consist of:

- A script to load the model and return predictions for submitted data.
- An environment in which the script will be run.

Creating an Entry Script (or scoring script)

It is a py file that must contain

- init(): Called when the service is initialized.
- run(raw_data): Called when new data is submitted to the service.

```
import json
import joblib
import numpy as np
from azureml.core.model import Model

# Called when the service is loaded
def init():
    global model
    # Get the path to the registered model file and load it
    model_path = Model.get_model_path('classification_model')
    model = joblib.load(model_path)

# Called when a request is received
def run(raw_data):
    # Get the input data as a numpy array
    data = np.array(json.loads(raw_data)['data'])
    # Get a prediction from the model
    predictions = model.predict(data)
    # Return the predictions as any JSON serializable format
    return predictions.tolist()
```

Creating an Environment

You can use **CondaDependencies**

```
from azureml.core.conda_dependencies import CondaDependencies

# Add the dependencies for your model
myenv = CondaDependencies()
myenv.add_conda_package("scikit-learn")

# Save the environment config as a .yml file
env_file = 'service_files/env.yml'
with open(env_file,"w") as f:
    f.write(myenv.serialize_to_string())
print("Saved dependency info in", env_file)
```

Combining the Script and Environment in an InferenceConfig

```
from azureml.core.model import InferenceConfig

classifier_inference_config = InferenceConfig(runtime= "python",
                                source_directory = 'service_files',
                                entry_script="score.py",
                                conda_file="env.yml")
```

3. Define a Deployment Configuration

Now, select the compute target to deploy to.

OBS: if deploying to AKS, create the cluster and a compute target for it before deploying.

```
from azureml.core.compute import ComputeTarget, AksCompute

cluster_name = 'aks-cluster'
compute_config = AksCompute.provisioning_configuration(location='eastus')
production_cluster = ComputeTarget.create(ws, cluster_name, compute_config)
production_cluster.wait_for_completion(show_output=True)
```

With the compute target created, define the deployment config

```
from azureml.core.webservice import AksWebservice

classifier_deploy_config = AksWebservice.deploy_configuration(cpu_cores = 1,
                                        memory_gb = 1)
```

The code to configure an ACI deployment is similar, except that you do not need to explicitly create an ACI compute target, and you must use the deploy_configuration class from the **azureml.core.webservice.AciWebservice** namespace. Similarly, you can use the **azureml.core.webservice.LocalWebservice** namespace to configure a local Docker-based service.

4. Deploy the Model

```
from azureml.core.model import Model

model = ws.models['classification_model']
service = Model.deploy(workspace=ws,
                       name = 'classifier-service',
                       models = [model],
                       inference_config = classifier_inference_config,
                       deployment_config = classifier_deploy_config,
                       deployment_target = production_cluster)
service.wait_for_deployment(show_output = True)
```

For ACI or local services, you can omit the deployment_target parameter (or set it to None).

Consuming a real-time inferencing service

Using the Azure Machine Learning SDK

For testing, you can use the AML SDK

```
import json

# An array of new data cases
x_new = [[0.1,2.3,4.1,2.0],
        [0.2,1.8,3.9,2.1]]

# Convert the array to a serializable list in a JSON document
json_data = json.dumps({"data": x_new})

# Call the web service, passing the input data
response = service.run(input_data = json_data)

# Get the predictions
predictions = json.loads(response)

# Print the predicted class for each case.
for i in range(len(x_new)):
    print (x_new[i]), predictions[i] )
```

Using a REST Endpoint

You can retrieve the service endpoint via the UI or the SDK:

```
endpoint = service.scoring_uri
print(endpoint)
import requests
import json

# An array of new data cases
x_new = [[0.1,2.3,4.1,2.0],
        [0.2,1.8,3.9,2.1]]

# Convert the array to a serializable list in a JSON document
json_data = json.dumps({"data": x_new})

# Set the content type in the request headers
request_headers = { 'Content-Type':'application/json' }

# Call the service
response = requests.post(url = endpoint,
                         data = json_data,
                         headers = request_headers)

# Get the predictions from the JSON response
predictions = json.loads(response.json())

# Print the predicted class for each case.
for i in range(len(x_new)):
    print (x_new[i]), predictions[i] )
```

Authentication

There are two kinds of auth

- **Key**: Requests are authenticated by specifying the key associated with the service.

- **Token**: Requests are authenticated by providing a JSON Web Token (JWT).

OBS: By default, authentication is disabled for ACI services, and set to key-based authentication for AKS services (for which primary and secondary keys are automatically generated). You can optionally configure an AKS service to use token-based authentication (which is not supported for ACI services).
You can retrieve the keys for a **WebService** as

```
primary_key, secondary_key = service.get_keys()
```

To use a token, the application needs to use a service-principal auth to verity the identity through AAD and call the **get_token** method to create a time-limited token.

```
import requests
import json

# An array of new data cases
x_new = [[0.1,2.3,4.1,2.0],
        [0.2,1.8,3.9,2.1]]

# Convert the array to a serializable list in a JSON document
json_data = json.dumps({"data": x_new})

# Set the content type in the request headers
request_headers = { "Content-Type":"application/json",
                    "Authorization":"Bearer " + key_or_token }

# Call the service
response = requests.post(url = endpoint,
                         data = json_data,
                         headers = request_headers)

# Get the predictions from the JSON response
predictions = json.loads(response.json())

# Print the predicted class for each case.
for i in range(len(x_new)):
    print (x_new[i]), predictions[i] )
```

Troubleshooting service deployment

Check the Service State

```
from azureml.core.webservice import AksWebservice

# Get the deployed service
service = AciWebservice(name='classifier-service', workspace=ws)

# Check its state
print(service.state)
```

OBS: To view the state of a service, you must use the compute-specific service type (for example AksWebservice) and not a generic WebService object.

Review Service Logs

```
print(service.get_logs())
```

Deploy to a Local Container

A quick check on runtime errors can be done by deploying to a local container.

```
from azureml.core.webservice import LocalWebservice

deployment_config = LocalWebservice.deploy_configuration(port=8890)
service = Model.deploy(ws, 'test-svc', [model], inference_config, deployment_config)
```

You can then test the locally deployed service using the SDK `service.run(input_data = json_data)` and troubleshoot runtime issues by making changes to the scoring file and reloading the service without redeploying (this can ONLY be done with a local service)

```
service.reload()
print(service.run(input_data = json_data))
```

Knowledge Check

1. You've trained a model using the Python SDK for Azure Machine Learning. You want to deploy the model as a containerized real-time service with high scalability and security. What kind of compute should you create to host the service?

    - An Azure Kubernetes Services (AKS) inferencing cluster.
    - A compute instance with GPUs.
    - A training cluster with multiple nodes.

    Answer

    You should use an AKS cluster to deploy a model as a scalable, secure, containerized service.

2. You're deploying a model as a real-time inferencing service. What functions must the entry script for the service include?

    - main() and score()
    - base() and train()
    - init() and run()

    Answer

    You must implement init and run functions in the entry (scoring) script.

Automate machine learning model selection with Azure Machine Learning

Show content

Learning objectives

OBS: Azure Machine Learning includes support for automated machine learning through a visual interface in Azure Machine Learning studio for Enterprise edition workspaces only. SDK is enabled in both Basic and Enterprise.

- Use Azure Machine Learning's automated machine learning capabilities to determine the best performing algorithm for your data.
- Use automated machine learning to preprocess data for training.
- Run an automated machine learning experiment.

Automated machine learning tasks and algorithms

You can automate classification, regression and time series forecasting.

There is a huge list of supporting algorithms for each task. By default, automated ML will randomly select from the full range of algorithms, but you can block individual algorithms.

Preprocessing and featurization

As well as trying the algorithms, automated ML can also apply preprocessing to the data to improve the performance.

- **Scaling and Normalization**: they are applied automatically to prevent any large numeric feature to dominate the taining.
- **Optional Featurization**: You can choose to apply preprocessing such as:
    - Missing value imputation
    - Categorical encoding
    - Dropping high cardinality features (as IDs)
    - Feature engineering (e.g., deriving individual date parts from DateTime features)

More information here.

Running automated machine learning experiments

You can use the UI (Enterprise) or the SDK.

Configuring an Automated Machine Learning Experiment

With the SDK you have greater flexibility and you can set experiment options using the **AutoMLConfig** class:

```
from azureml.train.automl import AutoMLConfig

automl_run_config = RunConfiguration(framework='python')
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)
```

Specifying Data for Training

With the UI, you can just select the training **dataset**. With the SDK, you can submit the data in the following ways:

- Specify a dataset or dataframe of training data that includes features and the label to be predicted.
- Specify a dataset, dataframe, or numpy array of X values containing the training features, with a corresponding y array of label values.

For both cases, you can optionally specify a validation dataset that will be used to validate the model. If it is not provided, Cross-Validation will be applied.

Specifying the Primary Metric

One of the most important settings. You can get all the metrics for a particular task as follows:

```
from azureml.train.automl.utilities import get_primary_metrics

get_primary_metrics('classification')
```

You can find a full list of primary metrics here.

Submitting an Automated Machine Learning Experiment

Automated ML experiments are submitted as any other experiment with the SDK:

```
from azureml.core.experiment import Experiment

automl_experiment = Experiment(ws, 'automl_experiment')
automl_run = automl_experiment.submit(automl_config)
```

You can monitor the runs in AML Studio or in the Jupyter Notebooks **RunDetails** widget.

Retrieving the Best Run and its Model

```
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)
```

Exploring Preprocessing Steps

AutoML uses SKlearn pipelines to encapsulate the processing steps. You view those steps in the fittedmodel obtained from the best run as shown below:

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

Knowledge Check

1. You are using automated machine learning to train a model that predicts the species of an iris based on its petal and sepal measurements. Which kind of task should you specify for automated machine learning?

    - Regression
    - Forecasting
    - Classification

    Answer

    Predicting a class requires a classification task.

2. You have submitted an automated machine learning run using the Python SDk for Azure Machine Learning. When the run completes, which method of the run object should you use to retrieve the best model?

    - get_output()
    - load_model()
    - get_metrics()

    Answer

    The get_output method of an automated machine learning run returns the best mode and the child run that trained it.

Get started with Machine Learning with an Azure Data Science Virtual Machine

Introduction to the Azure Data Science Virtual Machine (DSVM)

Show content

Learning objectives

- Learn about the types of Data Science Virtual Machines
- Learn what type of DSVM to use for each type of use case

When to use an Azure DSVM?

Azure DSVM makes it easy to maintain consistency in the evolving Data Science environments.

It also provides samples in Jupyter Notebooks and scripts for Python and R to learn about Microsoft and Azure ML services:

- How to connect to cloud Datastores with Azure ML and how to build models.
- Deep Learning samples using Microsoft Cognitive Services.
- How to compare Microsoft R and open source R and how to operationalize models with ML Services in SQL Server.

Types of Azure DSVM

- **Windows vs. Linux**: Windows Server 2012 and 2016 vs. Ubuntu 16.04 LTS and CentOS 7.4
- **Deep Learning**: The Deep Learning DSVM comes preconfigured and preinstalled with many tools and you can select high-speed GPU based machines.
- **Geo AI DSVM**: VM optimized for geospatial and location data. It has ArcGIS Pro system integrated.

Use cases for a DSVM

- **Collaborate as a team using DSVMs**: Working with cloud-based resources that can share the same configuration helps to ensure that all team members have a consistent development environment.
- **Address issues with DSVMs**: As issues related to environment mismatches are reduced. Giving DSVMs to students in a class.
- **Use on-demand elastic capacity for large-scale projects**: As it helps to replicate data science environments on demand to allow high-powered computing resources to be run.
- **Experiment and evaluate on a DSVM**: As they are easy to create, they can be used for demos and short experiments.
- **Learn about DSVMs and deep learning**: The flexibility of the underlying compute power (scaling or switching to GPU) makes it easy to train all kind of models.

Knowledge Check

1. Which of the following is a reason to use an Azure Data Science Virtual Machine?

    - You want to create an Azure Databricks workspace.
    - You want to get a jump-start on data science work.
    - You want to deploy a web application to it.

    Answer

    The purpose of Data Science Virtual Machines is to give a data scientist the tools they need, pre-installed, and ready to go.

2. Which of the following is installed on a Data Science Virtual Machine?

    - Azure Data Warehouse
    - Jupyter Notebook
    - Azure Machine Learning Studio

    Answer

    Jupyter Notebook is installed on Data Science Virtual Machines and provides a great data science development tool.

Explore the types of Azure Data Science Virtual Machines

Show content

Learning objectives

- Learn how to create Windows-based and Linux-based DSVMs
- Explore the Deep Learning Data Science Virtual Machines
- Work with Geo AI Data Science Virtual Machines

Windows-Based DSVMs

You can use the Windows-based DSVM to jump-start your data science projects. You don't pay for the DSVM image, just usage fees.

The image comes with a bunch of features:

- Tutorials
- Support for Office
- SQL Server integrated with ML Services
- Preinstalled languages: R, Python, SQL, C#
- Data Science tools such as Azure ML SDK for Python, Anaconda, Jupyter...

- ML tools as Azure Congitive Services support, H2O, Tensorflow, Weka...

Deep Learning Virtual Machine

Deep Learning Virtual Machines (DLVMs) use GPU-based hardware that provide increased mathematical calculation speed for faster model training. The image can be either Windows or Ubuntu.

The DLVM simplifies the tool selection process by including preconfigured tools for different situations.

Geo AI Data Science VM with ArcGIS

Both Python and R work with ArcGIS Pro, and are preconfigured on the Geo AI Data Science VM.

The image includes a large set of tools as DL frameworks, Keras, Caffe2 and Spark standalone.

OBS: Tools need to be compatible with GPUs.
It also comes bundled with IDEs such as visual studio or PyCharm.

Examples of Geo AI include:

- Real-time results of traffic conditions
- Driver availability in Uber or Lyft at any time
- Deep learning for disaster response
- Urban growth prediction

Knowledge Check

1. You want to learn about how to use Azure services related to machine learning with as little fuss as possible installing and configuring software and locating demonstration scripts. Which Data Science Virtual Machine type would best suit these needs?

    - Deep Learning DSVM
    - Windows 2016 DSVM
    - Geo AI Data Science VM with ArcGIS DSVM

    Answer

    The Windows 2016 gives you the most popular data science tools installed and configured and includes many sample scripts for using Azure machine learning related services.

2. You need to train deep learning models to do image recognition using a lot of training data in the form of images. Which DSVM configuration would be best for the fastest model training?

    - Windows 2016 with standard CPUs.
    - Geo AI Data Science VM with ArcGIS DSVM
    - Deep Learning VM which is configured to use GPUs.

    Answer

The DSVM includes all the software needed for training deep learning models and use graphic processor units (GPUs) which perform calculations much faster than standard CPUs.

Provision and use an Azure Data Science Virtual Machine

Show content

This module is based on exercise, so it's best followed here.

Knowledge Check

1. What method did we use to log into a Windows-Based Data Science VM?

    - Remote Desktop Protocol (RDP)
    - HTTP
    - ODBC

    Answer

    RDP: A step by step walk through explains all the steps to connect to a Windows-based DSVM.

2. What development environment has pre-loaded sample code available?

    - PyCharm
    - Zeppelin Notebook
    - Jupyter Notebook

    Answer

    Jupyter: We showed that many sample notebooks are installed that demonstrate how to use Microsoft Machine Learning technologies.

3. What type of Jupyter Notebook cell is used to provide annotations?

    - Code cell
    - Markdown cell
    - Raw cell

    Answer

    Markdown support rich formatting and is ideal for adding comments and annotations to your notebooks.