

Contents

Azure Machine Learning Documentation

Overview

[What is Azure Machine Learning service?](#)

[How the service works](#)

[What's happening to Workbench?](#)

[Release notes](#)

Quickstarts

[Get started - Portal](#)

[Get started - Python](#)

Tutorials

[Image classification \(MNIST data\)](#)

[1. Train a model](#)

[2. Deploy a model](#)

[Regression \(NYC Taxi data\)](#)

[1. Prepare data for modeling](#)

[2. Auto-train an ML model](#)

Samples

Concepts

[Automated machine learning](#)

[Model management](#)

[Accelerate with FPGAs](#)

[Machine learning pipelines](#)

How-to guides

[Manage workspaces](#)

[Set up development environment](#)

[Configure your dev environment](#)

[Visual Studio Code extension](#)

[Get started](#)

[Train and deploy](#)

[Prepare data](#)

[Load data](#)

[Transform data](#)

[Write data](#)

[Train models](#)

[Set up training environments](#)

[Access data in training](#)

[Create estimators in training](#)

[Use PyTorch](#)

[Use TensorFlow](#)

[Tune hyperparameters](#)

[Track experiments and metrics](#)

[Automate machine learning](#)

[Configure auto training](#)

[Use remote compute targets](#)

[ONNX models](#)

[Create and deploy](#)

[Deploy models](#)

[Where to deploy models](#)

[FPGAs](#)

[Troubleshoot](#)

[Secure with SSL](#)

[Consume web services](#)

[Consume in real-time](#)

[Run batch predictions](#)

[Monitor web services](#)

[Collect and evaluate model data](#)

[Monitor with Application Insights](#)

[Create your first pipeline](#)

[Manage resource quotas](#)

[Use the Machine Learning CLI](#)

[Reference](#)

[Machine learning SDK](#)

[Data prep SDK](#)

[Monitoring SDK](#)

[Resources](#)

[Azure roadmap](#)

[Pricing](#)

[Regional availability](#)

[Known issues](#)

[Get support](#)

[Export and delete data](#)

[Deprecated docs](#)

[Migrate from Workbench](#)

[Export or delete account data \(deprecated\)](#)

[Workbench quickstart \(deprecated\)](#)

[Tutorial 1: Prepare data](#)

[Tutorial 2: Build models](#)

[Tutorial 3: Deploy models](#)

[Tutorial 4: Advanced data preparation](#)

[Model Management overview](#)

[Top CLI tasks](#)

[Use Workbench](#)

[Work with Python IDEs](#)

[Use Jupyter notebooks](#)

[Integrate with Git repos](#)

[Structure projects with TDSP](#)

[Roaming and collaboration](#)

[Use IDE extensions](#)

[Use Visual Studio Tools for AI](#)

[Use Visual Studio Code Tools for AI](#)

[Use Azure IoT Edge AI Toolkit](#)

[Configure compute environment](#)

[Configure experimentation](#)

[Create DSVM and HDI Spark cluster](#)

Train with GPUs

Acquire and understand data

Add data source

Data preparation guide

Advanced data preparation

Data preparation transformations

Combine columns by example

Derive column by example

Split column by example

Expand JSON

Develop models

Classify Iris using CLI

Track run history and model metrics

Find run with the best accuracy

Read and write files

Use MMLSpark library

Deploy & consume models

Set up Model Management

Scale clusters and services

Enable SSL on an MLC cluster

Deploy a web service

Customize a container

Deploy to an IoT Edge device

Consume a web service

Collect model data

Deployment troubleshooting guide

Old examples

Document collection analysis

Q & A matching

Predictive maintenance

Aerial image classification

Server workload forecasting on terabytes of data

- [Energy demand time series forecasting](#)
- [Distributed tuning of hyperparameters](#)
- [Customer churn prediction](#)
- [Sentiment analysis with deep learning](#)
- [Biomedical entity recognition - TDSP project](#)
- [Classify US incomes - TDSP project](#)
- [Image classification using CNTK](#)
- [Deep Learning for Predictive Maintenance](#)
- [Old Reference](#)
 - [Python extensions for data preparation](#)
 - [Supported data sources](#)
 - [Load Azure Cosmos DB as data source](#)
 - [Supported inspectors](#)
 - [Supported data destinations](#)
 - [Supported execution and data environment combinations](#)
 - [Sample of filter expressions in Python](#)
 - [Sample of custom data flow transforms in Python](#)
 - [Sample of source connections in Python](#)
 - [Sample of destination connections in Python](#)
 - [Sample of column transforms in Python](#)
 - [Data preparation execution API](#)
 - [Experimentation Service configuration](#)
 - [Logging API](#)
 - [Experimentation Service template](#)
 - [Model Management API](#)
 - [Model Management CLI](#)
 - [Model data collection API](#)
- [Domain packages](#)
 - [Computer vision package deprecated](#)
 - [Package install](#)
 - [Model and deploy](#)
 - [Image classification](#)

[Object detection](#)

[Image similarity](#)

[Improve model accuracy](#)

[Forecasting package deprecated](#)

[Overview and install](#)

[Build and deploy forecast models](#)

[FPGA acceleration package](#)

[Text analytics package](#)

[Overview and install deprecated](#)

[Build and deploy text classification models](#)

[FAQs](#)

[Old Issues](#)

What is Azure Machine Learning service?

12/11/2018 • 4 minutes to read • [Edit Online](#)

Azure Machine Learning service is a cloud service that you can use to train, deploy, automate and manage machine learning models, all at the broad scale that the cloud provides.

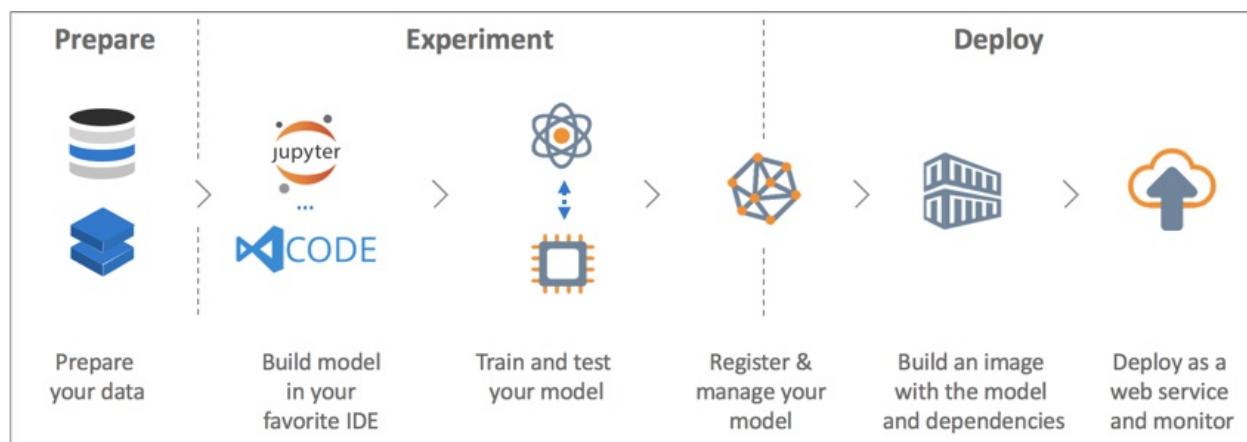
What is machine learning?

Machine learning is a data science technique that allows computers to use existing data to forecast future behaviors, outcomes, and trends. Using machine learning, computers learn without being explicitly programmed.

Forecasts or predictions from machine learning can make apps and devices smarter. For example, when you shop online, machine learning helps recommend other products you might like based on what you've purchased. Or when your credit card is swiped, machine learning compares the transaction to a database of transactions and helps detect fraud. And when your robot vacuum cleaner vacuums a room, machine learning helps it decide whether the job is done.

What is Azure Machine Learning service?

Azure Machine Learning service provides a cloud-based environment you can use to develop, train, test, deploy, manage, and track machine learning models.



Azure Machine Learning service fully supports open-source technologies, so you can use tens of thousands of open-source Python packages with machine learning components such as TensorFlow and scikit-learn. Support for rich tools, such as [Jupyter notebooks](#) or the [Azure Machine Learning for Visual Studio Code extension](#), makes it easy to interactively explore data, transform it, and then develop, and test models. Azure Machine Learning service also includes features that [automate model generation and tuning](#) to help you create models with ease, efficiency, and accuracy.

Azure Machine Learning service lets you start training on your local machine, and then scale out to the cloud. With many available [compute targets](#) such as Azure Machine Learning Compute and [Azure Databricks](#), and with [advanced hyperparameter tuning services](#), you can build better models faster, using the power of the cloud.

When you have the right model, you can easily deploy it in a container such as Docker. This means that it's simple to deploy to Azure Container Instances or Azure Kubernetes Service, or you can use the container in your own deployments, either on-premises or in the cloud. For more information, see the [How to deploy and where](#) document. You can manage the deployed models, and track multiple runs as you experiment to find the best solution. Once deployed, your model can return predictions in [real-time](#), or [asynchronously](#) on large quantities of data.

And with advanced [machine learning pipelines](#), you can collaborate on all the steps of data preparation, model training and evaluation, and deployment.

What can I do with Azure Machine Learning service?

Azure Machine Learning service can auto-train a model and auto-tune it for you. For an example, see [Tutorial: Automatically train a classification model with Azure Automated Machine Learning](#).

Using the Azure Machine Learning [SDK](#) for Python, along with open-source Python packages, you can build and train highly accurate machine learning and deep learning models yourself in an Azure Machine Learning service workspace. You can choose from many machine learning components available in open-source Python packages, such as the following:

- [Scikit-learn](#)
- [Tensorflow](#)
- [PyTorch](#)
- [CNTK](#)
- [MXNet](#)

Once you have a model, you use it to create a container (such as Docker) that can be deployed locally for testing. Once testing is done, the model can be deployed as a production web service in either Azure Container Instances or Azure Kubernetes Service. For more information, see the [How to deploy and where](#) document.

You then can manage your deployed models using the [Azure Machine Learning SDK for Python](#) or the [Azure portal](#). You can evaluate model metrics, retrain, and redeploy new versions of the model, all while tracking the model's experiments.

To get started using Azure Machine Learning service, see [Next steps](#) below.

How is Azure Machine Learning service different from Studio?

Azure Machine Learning Studio is a collaborative, drag-and-drop visual workspace where you can build, test, and deploy machine learning solutions without needing to write code. It uses pre-built and pre-configured machine learning algorithms and data-handling modules.

Use Machine Learning Studio when you want to experiment with machine learning models quickly and easily, and the built-in machine learning algorithms are sufficient for your solutions.

Use Machine Learning service if you work in a Python environment, you want more control over your machine learning algorithms, or you want to use open-source machine learning libraries.

NOTE

Models created in Azure Machine Learning Studio cannot be deployed or managed by Azure Machine Learning service.

Free trial

If you don't have an Azure subscription, create a free account before you begin. Try the [free or paid version of Azure Machine Learning service](#) today.

You get credits to spend on Azure services. After they're used up, you can keep the account and use [free Azure services](#). Your credit card is never charged unless you explicitly change your settings and ask to be charged.

Alternatively, you can [activate MSDN subscriber benefits](#), which gives you credits every month that you can use for paid Azure services.

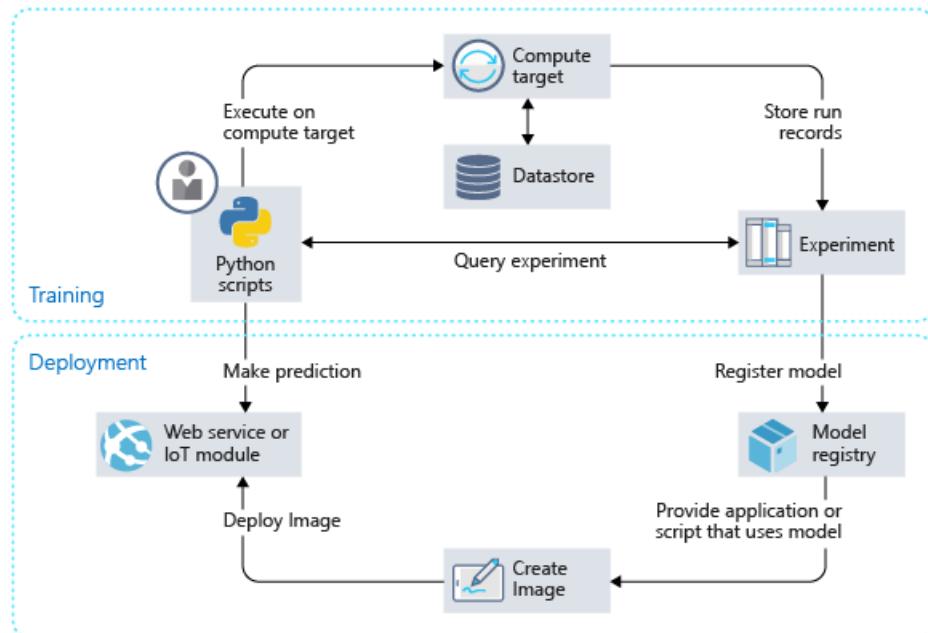
Next steps

- Create a Machine Learning Service Workspace to get started [using Azure portal](#) or in Python.
- Follow the full-length tutorial, [Train and deploy an image classification model with Azure Machine Learning](#).
- [Use Azure Machine Learning to auto-generate and autotune a model](#).
- Learn about [machine learning pipelines](#) to build, optimize, and manage your machine learning scenarios.
- Read the in-depth [Azure Machine Learning service architecture and concepts](#) article.
- For more information on other machine learning products from Microsoft, see [Other machine learning products from Microsoft](#).

How the Azure Machine Learning service works: architecture and concepts

12/10/2018 • 10 minutes to read • [Edit Online](#)

This document describes the architecture and concepts for the Azure Machine Learning service. The following diagram shows the major components of the service, and illustrates the general workflow when using the service:



The workflow generally follows these steps:

1. Develop machine learning training scripts in **Python**.
2. Create and configure a **compute target**.
3. **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**.
4. **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.
5. Once a satisfactory run is found, register the persisted model in the **model registry**.
6. Develop a scoring script.
7. **Create an Image** and register it in the **image registry**.
8. **Deploy the image** as a **web service** in Azure.

NOTE

While this document defines terms and concepts used by Azure Machine Learning, it does not define terms and concepts for the Azure platform. For more information on Azure platform terminology, see the [Microsoft Azure glossary](#).

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. This information is used to determine which training run produces the best model.

Models are registered with the workspace. A registered model and scoring scripts are used to create an image. The image can then be deployed into Azure Container Instances, Azure Kubernetes Service, or to a field-programmable gate array (FPGA) as a REST-based HTTP endpoint. It can also be deployed to an Azure IoT Edge device as a module.

You can create multiple workspaces, and each workspace can be shared by multiple people. When sharing a workspace, control access to the workspace by assigning the following roles to users:

- Owner
- Contributor
- Reader

When you create a new workspace, it automatically creates several Azure resources that are used by the workspace:

- [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 models.
- [Azure Key Vault](#) - Stores secrets used by compute targets and other sensitive information needed by the workspace.

NOTE

Instead of creating new versions, you can also use existing Azure services.

The following diagram is a taxonomy of the workspace:

Model

At its simplest, a model is a piece of code that takes an input and produces output. Creating a machine learning model involves selecting an algorithm, providing it with data, and tuning hyperparameters. Training is an iterative process that produces a trained model, which encapsulates what the model learned during the training process.

A model is produced by a run in Azure Machine Learning. You can also use a model trained outside of Azure Machine Learning. A model can be registered under an Azure Machine Learning service workspace.

Azure Machine Learning Service is framework agnostic. You can use any popular machine learning framework when creating a model, such as scikit-learn, xgboost, PyTorch, TensorFlow, Chainer, and CNTK.

For an example of training a model, see the [Quickstart: Create a machine learning Service workspace](#) document.

Model registry

The model registry keeps track of all the models in your Azure Machine Learning service workspace.

Models are identified by name and version. Each time you register a model with the same name as an existing one, the registry assumes that it is a new version. The version is incremented and the new model is registered under the name.

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.

For an example of registering a model, see the [Train an image classification model with Azure Machine Learning](#) document.

Image

Images provide a way to reliably deploy a model, along with all components needed to use the model. An image contains the following items:

- A model.
- A scoring script or application. This script is used to pass input to the model and return the output of the model.
- Dependencies needed by the model or scoring script/application. For example, you might include a Conda environment file that lists Python package dependencies.

There are two types of images that can be created by Azure Machine Learning:

- FPGA image: Used when deploying to a field-programmable gate array in the Azure cloud.
- Docker image: Used when deploying to compute targets other than FPGA. For example, Azure Container Instances and Azure Kubernetes Service.

For an example of creating an image, see the [Deploy an image classification model in Azure Container Instance](#) document.

Image registry

The image registry keeps track of images created from your models. You can provide additional metadata tags when creating the image. Metadata tags are stored by the image registry and can be queried to find your image.

Deployment

A deployment is an instantiation of your image into either a Web Service that may be hosted in the cloud or an IoT Module for integrated device deployments.

Web service

A deployed web service can use Azure Container Instances, Azure Kubernetes Service, or field-programmable gate arrays (FPGA). The service is created from an image that encapsulates your model, script, and associated files. The image has a load-balanced, HTTP endpoint that receives scoring requests sent to the web service.

Azure helps you monitor your Web service deployment by collecting Application Insight telemetry and/or model telemetry if you have chosen to enable this feature. The telemetry data is only accessible to you, and stored in your Application Insights and storage account instances.

If you have enabled automatic scaling, Azure will automatically scale your deployment.

For an example of deploying a model as a web service, see the [Deploy an image classification model in Azure Container Instance](#) document.

IoT Module

A deployed IoT Module is a Docker container that includes your model and associated script or application and any additional dependencies. These modules are deployed using Azure IoT Edge on edge devices.

If you have enabled monitoring, Azure collects telemetry data from the model inside the Azure IoT Edge module. The telemetry data is only accessible to you, and stored in your storage account instance.

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

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.

Run

A run is a record that contains the following information:

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

A run is produced when you submit a script to train a model. A run can have zero or more child runs. So the top-level run might have two child runs, each of which may have their own child runs.

For an example of viewing runs produced by training a model, see the [Quickstart: Get started with Azure Machine Learning service](#) document.

Experiment

An experiment is a grouping of many runs from a given script. It always belongs to a workspace. When you submit a run, you provide an experiment name. Information for the run is stored under that experiment. If you submit a run and specify an experiment name that doesn't exist, a new experiment with that name is automatically created.

For an example of using an experiment, see the [Quickstart: Get started with Azure Machine Learning service](#) document.

Pipeline

Machine learning pipelines are used to create and manage workflows that stitch together machine learning phases. For example, a pipeline might include data preparation, model training, model deployment, and inferencing phases. Each phase can encompass multiple steps, each of which can run unattended in various compute targets.

For more information on machine learning pipelines with this service, see the article [Pipelines and Azure Machine Learning](#).

Compute target

A compute target is the compute resource used to run your training script or host your service deployment. The supported compute targets are:

COMPUTE TARGET	TRAINING	DEPLOYMENT
Your local computer	✓	
Azure Machine Learning Compute	✓	

COMPUTE TARGET	TRAINING	DEPLOYMENT
A Linux VM in Azure (such as the Data Science Virtual Machine)	✓	
Azure Databricks	✓	
Azure Data Lake Analytics	✓	
Apache Spark for HDInsight	✓	
Azure Container Instance		✓
Azure Kubernetes Service		✓
Azure IoT Edge		✓
Project Brainwave (Field-programmable gate array)		✓

Compute targets are attached to a workspace. Compute targets other than the local machine are shared by users of the workspace.

Managed and unmanaged compute targets

Managed compute targets are created and managed by the Azure Machine Learning service. These compute targets are optimized for ML workloads. **Azure Machine Learning Compute** is the only managed compute target at this time (December 4th, 2018). Additional managed compute targets may be added in the future. ML Compute instances can be created directly through the workspace by using the Azure portal, Azure Machine Learning SDK, or Azure CLI. All other compute targets must be created outside the workspace, and then attached to it.

Unmanaged compute targets are not managed by the Azure Machine Learning service. You may need to create them outside Azure Machine Learning and then attach them to your workspace before use. These compute targets can require additional steps to maintain, or improve performance for ML workloads.

For information on selecting a compute target for training, see the [Select and use a compute target to train your model](#) document.

For information on selecting a compute target for deployment, see the [Deploy models with the Azure Machine Learning service](#) document.

Run configuration

A run configuration is a set of instructions that defines how a script should be run in a given compute target. It includes a wide set of behavior definitions, such as whether to use an existing Python environment or use a Conda environment built from specification.

A run configuration can be persisted into a file inside the directory that contains your training script, or constructed as an in-memory object and used to submit a run.

For example run configurations, see the [Select and use a compute target to train your model](#) document.

Training script

To train a model, you specify the directory that contains the training script and associated files. You also specify an experiment name, which is used to store information gathered during training. During training, the entire directory is copied to the training environment (compute target), and the script specified by the run configuration is started. A snapshot of the directory is also stored under the experiment in the workspace.

For an example, see [Create a workspace with Python](#)

Logging

When developing your solution, use the Azure Machine Learning Python SDK in your Python script to log arbitrary metrics. After the run, query the metrics to determine if the run produced the model you want to deploy.

Snapshot

When submitting a run, Azure Machine Learning compresses the directory that contains the script as a zip file and sends it to the compute target. The zip is then expanded and the script is run there. Azure Machine Learning also stores the zip file as a snapshot as part of the run record. Anyone with access to the workspace can browse a run record and download the snapshot.

Activity

An activity represents a long running operation. The following operations are examples of activities:

- Creating or deleting a compute target
- Running a script on a compute target

Activities can provide notifications through the SDK or Web UI so you can easily monitor the progress of these operations.

Next steps

Use the following links to get started using Azure Machine Learning:

- [What is Azure Machine Learning service](#)
- [Quickstart: Create a workspace with Python](#)
- [Tutorial: Train a model](#)

What is happening to Workbench in Azure Machine Learning service?

12/10/2018 • 5 minutes to read • [Edit Online](#)

The Workbench application and some other early features were deprecated and replaced in the September 2018 release to make way for an improved [architecture](#). The release contains many significant updates prompted by customer feedback to improve your experience. The core functionality from experiment runs to model deployment has not changed, but now you can use the robust [SDK](#) and [CLI](#) to accomplish your machine learning tasks and pipelines.

In this article, you'll learn about what changed and how it affects your pre-existing work with the Azure Machine Learning Workbench and its APIs.

What changed?

The latest release of Azure Machine Learning service includes:

- A [simplified Azure resources model](#)
- [New portal UI](#) to manage your experiments and compute targets
- A new, more comprehensive Python [SDK](#)
- A new expanded [Azure CLI extension](#) for machine learning

The [architecture](#) was redesigned with ease-of-use in mind. Instead of multiple Azure resources and accounts, you only need an [Azure Machine Learning service Workspace](#). You can create workspaces quickly in the [Azure portal](#). A workspace can be used by multiple users to store training and deployment compute targets, model experiments, Docker images, deployed models, and so on.

While there are new improved CLI and SDK clients in the current release, the desktop Workbench application itself is deprecated. Now, you can monitor your experiments in the [workspace dashboard in the Azure web portal](#). Use the dashboard to get your experiment history, manage the compute targets attached to your workspace, manage your models and Docker images, and even deploy web services.

How do I migrate?

Most of the artifacts created in the earlier version of the Azure Machine Learning service are stored in your own local or cloud storage. These artifacts won't ever disappear. To migrate, you need to register the artifacts again with the updated Azure Machine Learning service. Learn what you can migrate and how in this [migration article](#).

Support timeline

You can continue to use your experimentation and model management accounts as well as the Workbench application for a while longer after September 2018. Support for the following resources will be removed progressively in the 3-4 months after that release. You can still find the documentation for the old features in the [Resources section](#) at the bottom of the table of contents.

RETIREMENT PHASE

SUPPORT DETAILS FOR EARLIER FEATURES

RETIREMENT PHASE	SUPPORT DETAILS FOR EARLIER FEATURES
December 4, 2018	The ability to create <i>Azure Machine Learning Experimentation account</i> and <i>Model Management account</i> in the Azure portal and from the CLI has ended. The ability to create ML Compute Environments from the CLI has also ended. If you have an existing account, the CLI and the desktop Workbench continue to work in this phase.
January 9, 2019	Support for everything else, including the remaining APIs and the desktop Workbench ends on this date.

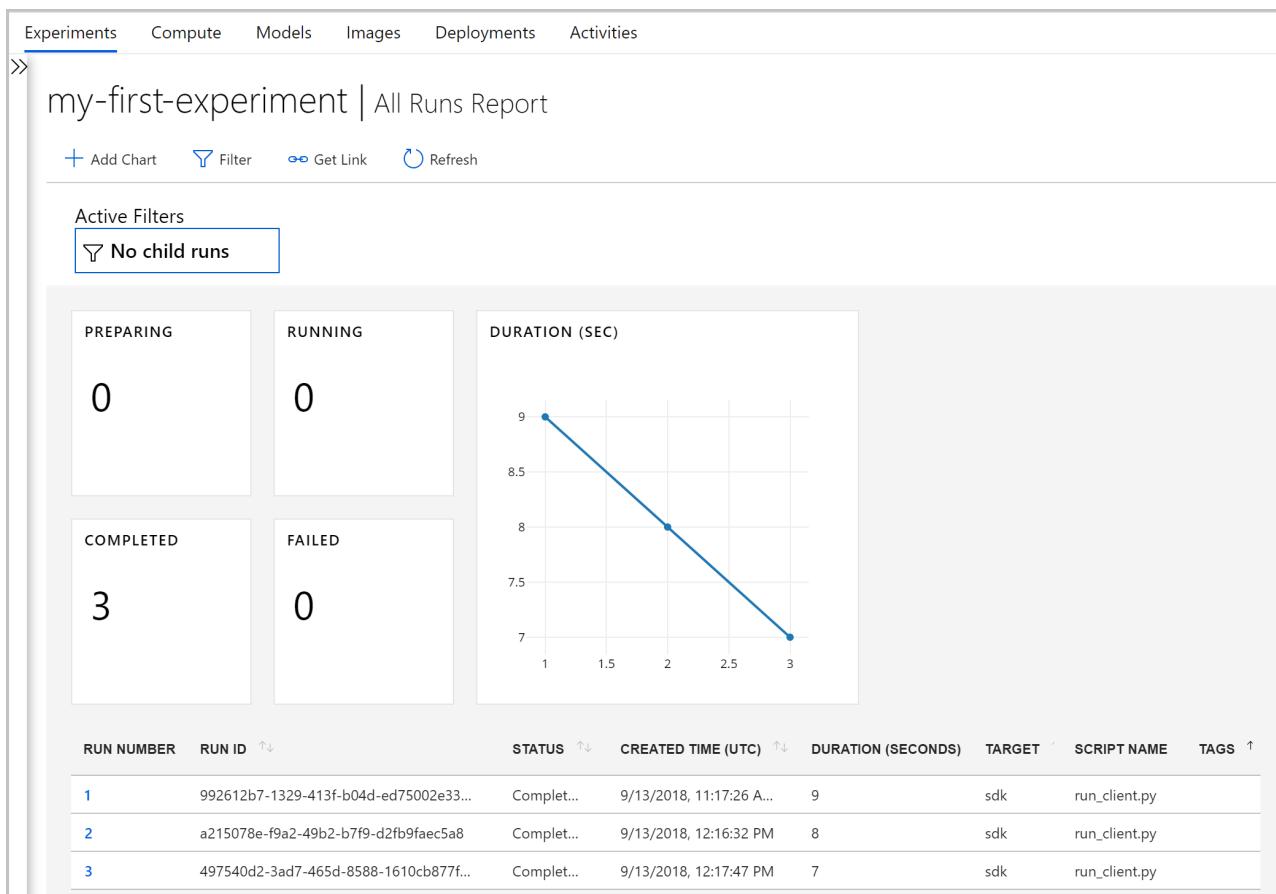
Start migrating today. All of the latest capabilities are available using the new [SDK](#), [CLI](#), and [portal](#).

What about run histories?

Run histories will remain accessible for a while. When you are ready to move to the updated version of Azure Machine Learning service, you can export these run histories if you want to keep a copy.

Run histories are now called *experiments* in the current release. You can collect your model's experiments and explore them using the SDK, CLI, or web portal.

The portal's workspace dashboard is supported on Edge, Chrome and Firefox browsers only.



Can I still prep data?

Your pre-existing data preparation files are not portable to the latest release since we don't have Workbench anymore. However, you can still prepare your data for modeling.

With smaller data sets, you can use the [Azure Machine Learning Data Prep SDK](#) to quickly prepare your data prior to modeling.

You can use this same [SDK](#) for larger data sets or use Azure Databricks to prepare big data sets.

Will projects persist?

You won't lose any code or work. In the older version, projects are cloud entities with a local directory. In the latest version, you attach local directories to the Azure Machine Learning service Workspace using a local config file. [See a diagram of the latest architecture](#).

Since much of the project content was already on your local machine, you just need to create a config file in that directory and reference it in your code to connect to your workspace. [Learn how migrate your existing projects](#).

Learn how to get started [in Python with the main SDK](#).

What about my registers models and images?

The models that you registered in your old model registry must be migrated to your new workspace if you want to continue to use them. You can do this by [downloading the models and re-registering them](#) in your new workspace.

The images that you created in your old image registry must be re-created in the new workspace to continue to use them. You can do this by following the [Configure and create image](#) sections.

What about deployed web services?

The models you deployed as web services using your Model Management account will continue to work for as long as Azure Container Service (ACS) is supported. Those web services will even work after support has ended for Model Management accounts. However, when support for the old CLI ends, so does your ability to manage those web services.

In the newer version, models are deployed as web services to Azure Container Instances(ACI) or Azure Kubernetes Service (AKS) clusters. You can also deploy to FPGAs and to the IoT edge. For more information, see the [How to deploy and where](#) document. Without having to change any of your scoring files, dependencies, and schemas, you can redeploy your models using the new SDK or CLI.

What about the old SDK & CLI?

Yes, they will continue to work until January (see the [timeline](#) above). We recommend that you start creating your new experiments and models with the latest SDK and/or CLI.

In the latest release, the new Python SDK allows you to interact with the Azure Machine Learning service in any Python environment. Learn how to install the latest [SDK](#). You can also use the [updated Azure CLI machine learning extension](#) with the rich set of `az ml` commands to interact the service in any command-line environment, including Azure portal cloud shell.

What about Azure Machine Learning for Visual Studio Code?

With this latest release, Azure Machine Learning for Visual Studio (VS) Code has been expanded and improved to work with the above new features.

The screenshot shows a Visual Studio Code window with the following details:

- File Explorer:** Shows a workspace named "MyTeamWorkspace" containing projects like "Experiments", "MNIST", "DiabetesDetection", "Compute", "Models", "Project MNIST", "Run Configs", "Docker", "GPU Cluster", and "Local".
- Code Editor:** The file "train.py" is open, showing Python code for training a neural network on the MNIST dataset.
- Terminal:** Shows the command "Azure: clauren@microsoft.com Anaconda, Inc. Python 3.6.5 (AzureML)".
- Output Panel:** Shows "Experiment MNIST" with tabs for Details, Outputs, Logs, and Snapshot. The "CHARTS" tab displays two line graphs: "training_acc" and "validation_acc".
- Status Bar:** Shows "Ln 1, Col 10" and other standard status bar information.

What about domain packages?

The domain packages for [Computer Vision](#), [Text Analytics](#), and [Forecasting](#) cannot be used with the latest version of Azure Machine Learning. However, you can still build and train computer vision, text, and forecasting models with latest Azure Machine Learning Python [SDK](#). To learn how to migrate pre-existing models built using the Computer Vision, Text Analytics, and Forecasting packages, contact us at AML-Packages@microsoft.com.

Next steps

Learn about [the latest architecture for the Azure Machine Learning service](#) and try one of the quickstarts or tutorials:

- [What is Azure Machine Learning service](#)
- [Quickstart: Create a workspace with Python](#)
- [Tutorial: Train a model](#)

Azure Machine Learning service release notes

12/10/2018 • 24 minutes to read • [Edit Online](#)

In this article, learn about the Azure Machine Learning service releases.

2018-12-04: General Availability

Azure Machine Learning service is now generally available.

Azure Machine Learning Compute

With this release, we are announcing a new managed compute experience through [Azure Machine Learning Compute](#). This compute can be used for Training and Batch inferencing, is single- to multi-node compute, and does the cluster management and job scheduling for the user. It autoscales by default, has support for both CPU and GPU resources and also allows using Low-Priority VMs for reduced cost. It replaces Batch AI compute for Azure Machine Learning.

Azure Machine Learning Compute can be created in Python, using Azure portal, or the CLI. It must be created in the region of your workspace, and cannot be attached to any other workspace. This compute uses a Docker container for your run, and packages your dependencies to replicate the same environment across all your nodes.

WARNING

We recommend creating a new workspace to use Azure Machine Learning Compute. There is a remote chance that users trying to create Azure Machine Learning Compute from an existing workspace might see an error. Existing compute in your workspace should continue to work unaffected.

Azure Machine Learning SDK for Python v1.0.2

- **Breaking changes**

- With this release, we are removing support for creating a VM from Azure Machine Learning. You can still attach an existing cloud VM or a remote on-premises server.
- We are also removing support for BatchAI, all of which should be supported through Azure Machine Learning Compute now.

- **New**

- For machine learning pipelines:
 - [EstimatorStep](#)
 - [HyperDriveStep](#)
 - [MpiStep](#)

- **Updated**

- For machine learning pipelines:
 - [DatabricksStep](#) now accepts runconfig
 - [DataTransferStep](#) now copies to and from a SQL datasource
 - Schedule functionality in SDK to create and update schedules for running published pipelines

Azure Machine Learning Data Prep SDK v0.5.2

- **Breaking changes**

- `SummaryFunction.N` was renamed to `SummaryFunction.Count`.

- **Bug Fixes**

- Use latest AML Run Token when reading from and writing to datastores on remote runs. Previously, if the AML Run Token is updated in Python, the Data Prep runtime will not be updated with the updated AML Run Token.
- Additional clearer error messages
- `to_spark_dataframe()` will no longer crash when Spark uses Kryo serialization
- Value Count Inspector can now show more than 1000 unique values
- Random Split no longer fails if the original Dataflow doesn't have a name

Docs and notebooks

- ML Pipelines
 - New and updated notebooks for getting started with pipelines, batch scoping, and style transfer examples: <https://aka.ms/aml-pipeline-notebooks>
 - Learn how to [create your first pipeline](#)
 - Learn how to [run batch predictions using pipelines](#)
- Azure Machine Learning compute
 - [Sample notebooks](#) are now updated to use this new managed compute.
 - [Learn about this compute](#)

Azure portal: new features

- Create and manage [Azure Machine Learning Compute](#) types in the portal.
- Monitor quota usage and [request quota](#) for Azure Machine Learning Compute.
- View Azure Machine Learning Compute cluster status in real-time.
- Virtual network support was added for Azure Machine Learning Compute and Azure Kubernetes Service creation.
- Re-run your published pipelines with existing parameters.
- New [automated machine learning charts](#) for classification models (lift, gains, calibration, feature importance chart with model explainability) and regression models (residuals and feature importance chart with model explainability).
- Pipelines can be viewed in Azure portal

2018-11-20

Azure Machine Learning SDK for Python v0.1.80

- **Breaking changes**

- `azureml.train.widgets` namespace has moved to `azureml.widgets`.
- `azureml.core.compute.AmlCompute` deprecates the following classes -
`azureml.core.compute.BatchAICompute` and `azureml.core.compute.DSVMCompute`. The latter class will be removed in subsequent releases. The AmlCompute class has an easier definition now, and simply needs a `vm_size` and the `max_nodes`, and will automatically scale your cluster from 0 to the `max_nodes` when a job is submitted. Our [sample notebooks](#) have been updated with this information and should give you examples on how to use this. We hope you like this simplification and lots of more exciting features to come in a later release!

Azure Machine Learning Data Prep SDK v0.5.1

Learn more about the Data Prep SDK by reading [reference docs](#).

- **New Features**

- Created a new DataPrep CLI to execute DataPrep packages and view the data profile for a dataset or dataflow

- Redesigned SetColumnType API to improve usability
- Renamed smart_read_file to auto_read_file
- Now includes skew and kurtosis in the Data Profile
- Can sample with stratified sampling
- Can read from zip files that contain CSV files
- Can split datasets row-wise with Random Split (e.g. into test-train sets)
- Can get all the column data types from a dataflow or a data profile by calling .dtypes
- Can get the row count from a dataflow or a data profile by calling .row_count

- **Bug Fixes**

- Fixed long to double conversion
- Fixed assert after any add column
- Fixed an issue with FuzzyGrouping, where it would not detect groups in some cases
- Fixed sort function to respect multi-column sort order
- Fixed and/or expressions to be similar to how Pandas handles them
- Fixed reading from dbfs path
- Made error messages more understandable
- Now no longer fails when reading on remote compute target using AML token
- Now no longer fails on Linux DSVM
- Now no longer crashes when non-string values are in string predicates
- Now handles assertion errors when Dataflow should fail correctly
- Now supports dbutils mounted storage locations on Azure Databricks

2018-11-05

Azure portal

The Azure portal for the Azure Machine Learning service has the following updates:

- A new **Pipelines** tab for published pipelines.
- Added support for attaching an existing HDInsight cluster as a compute target.

Azure Machine Learning SDK for Python v0.1.74

- **Breaking changes**

- *Workspace.compute_targets, datastores, experiments, images, models* and *webservices* are properties instead of methods. For example, replace *Workspace.compute_targets()* with *Workspace.compute_targets*.
- *Run.get_context* deprecates *Run.get_submitted_run*. The latter method will be removed in subsequent releases.
- *PipelineData* class now expects a datastore object as a parameter rather than *datastore_name*. Similarly, *Pipeline* accepts *default_datastore* rather than *default_datastore_name*.

- **New features**

- The Azure Machine Learning Pipelines [sample notebook](#) now uses MPI steps.
- The RunDetails widget for Jupyter notebooks is updated to show a visualization of the pipeline.

Azure Machine Learning Data Prep SDK v0.4.0

- **New features**

- Type Count added to Data Profile
- Value Count and Histogram is now available

- More percentiles in Data Profile
- The Median is available in Summarize
- Python 3.7 is now supported
- When you save a dataflow that contains datastores to a DataPrep package, the datastore information will be persisted as part of the DataPrep package
- Writing to datastore is now supported

- **Bug fixed**

- 64bit unsigned integer overflows are now handled properly on Linux
- Fixed incorrect text label for plain text files in smart_read
- String column type now shows up in metrics view
- Type count now is fixed to show ValueKinds mapped to single FieldType instead of individual ones
- Write_to_csv no longer fails when path is provided as a string
- When using Replace, leaving "find" blank will no longer fail

2018-10-12

Azure Machine Learning SDK for Python v0.1.68

- **New features**

- Multiple tenant support when creating new workspace.

- **Bugs fixed**

- The pynacl library version no longer needs to be pinned when deploying web service.

Azure Machine Learning Data Prep SDK v0.3.0

- **New features**

- Added method transform_partition_with_file(script_path), which allows users to pass in the path of a Python file to execute

2018-10-01

Azure Machine Learning SDK for Python v0.1.65

[Version 0.1.65](#) includes new features, more documentation, bug fixes, and more [sample notebooks](#).

See [the list of known issues](#) to learn about known bugs and workarounds.

- **Breaking changes**

- Workspace.experiments, Workspace.models, Workspace.compute_targets, Workspace.images, Workspace.web_services return dictionary, previously returned list. See [azureml.core.Workspace](#) API documentation.
- Automated Machine Learning removed normalized mean square error from the primary metrics.

- **HyperDrive**

- Various HyperDrive bug fixes for Bayesian, Performance improvements for get Metrics calls.
- Tensorflow 1.10 upgrade from 1.9
- Docker image optimization for cold start.
- Job's now report correct status even if they exit with error code other than 0.
- RunConfig attribute validation in SDK.
- HyperDrive run object supports cancel similar to a regular run: no need to pass any parameters.
- Widget improvements for maintaining state of drop-down values for distributed runs and HyperDrive

runs.

- TensorBoard and other log files support fixed for Parameter server.
- Intel(R) MPI support on service side.
- Bugfix to parameter tuning for distributed run fix during validation in BatchAI.
- Context Manager now identifies the primary instance.

- **Azure portal experience**

- `log_table()` and `log_row()` are supported in Run details.
- Automatically create graphs for tables and rows with 1,2 or 3 numerical columns and an optional categorical column.

- **Automated Machine Learning**

- Improved error handling and documentation
- Fixed run property retrieval performance issues.
- Fixed continue run issue.
- Fixed ensembling iteration issues.
- Fixed training hanging bug on MAC OS.
- Downsampling macro average PR/ROC curve in custom validation scenario.
- Removed extra index logic.
- Removed filter from `get_output API`.

- **Pipelines**

- Added a method `Pipeline.publish()` to publish a pipeline directly, without requiring an execution run first.
- Added a method `PipelineRun.get_pipeline_runs()` to fetch the pipeline runs which were generated from a published pipeline.

- **Project Brainwave**

- Updated support for new AI models available on FPGAs.

Azure Machine Learning Data Prep SDK v0.2.0

Version 0.2.0 includes following features and bugfixes:

- **New features**

- Support for one-hot encoding
- Support for quantile transform

- **Bug fixed:**

- Works with any Tornado version, no need to downgrade your Tornado version
- Value counts for all values, not just the top three

2018-09 (Public preview refresh)

A new, completely refreshed release of Azure Machine Learning: Read more about this release:

<https://azure.microsoft.com/blog/what-s-new-in-azure-machine-learning-service/>

Older notes: Sept 2017 - Jun 2018

2018-05 (Sprint 5)

With this release of Azure Machine Learning, you can:

- Featurize images with a quantized version of ResNet 50, train a classifier based on those features, and [deploy that model to an FPGA on Azure](#) for ultra-low latency inferencing.

- Quickly build and deploy highly-accurate machine learning and deep learning models using [custom Azure Machine Learning Packages](#) for the following domains:

- Computer vision
- Text analytics
- Forecasting

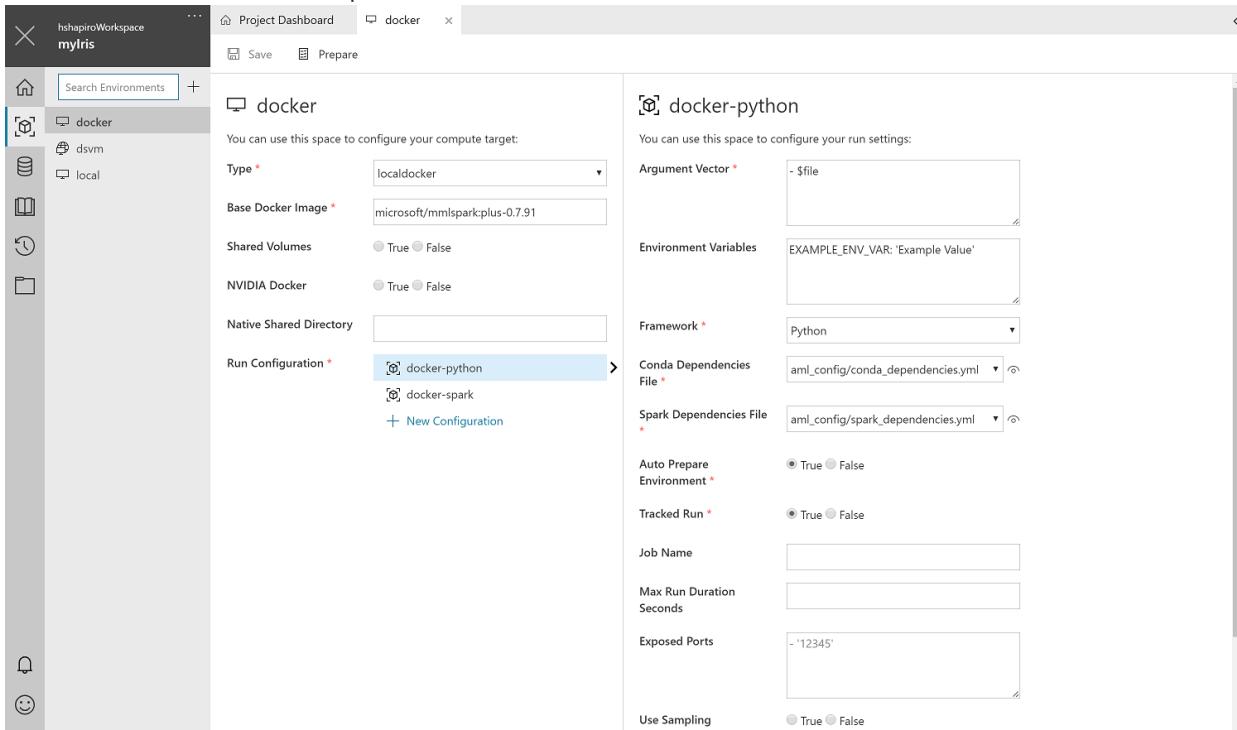
2018-03 (Sprint 4)

Version number: 0.1.1801.24353 ([Find your version](#))

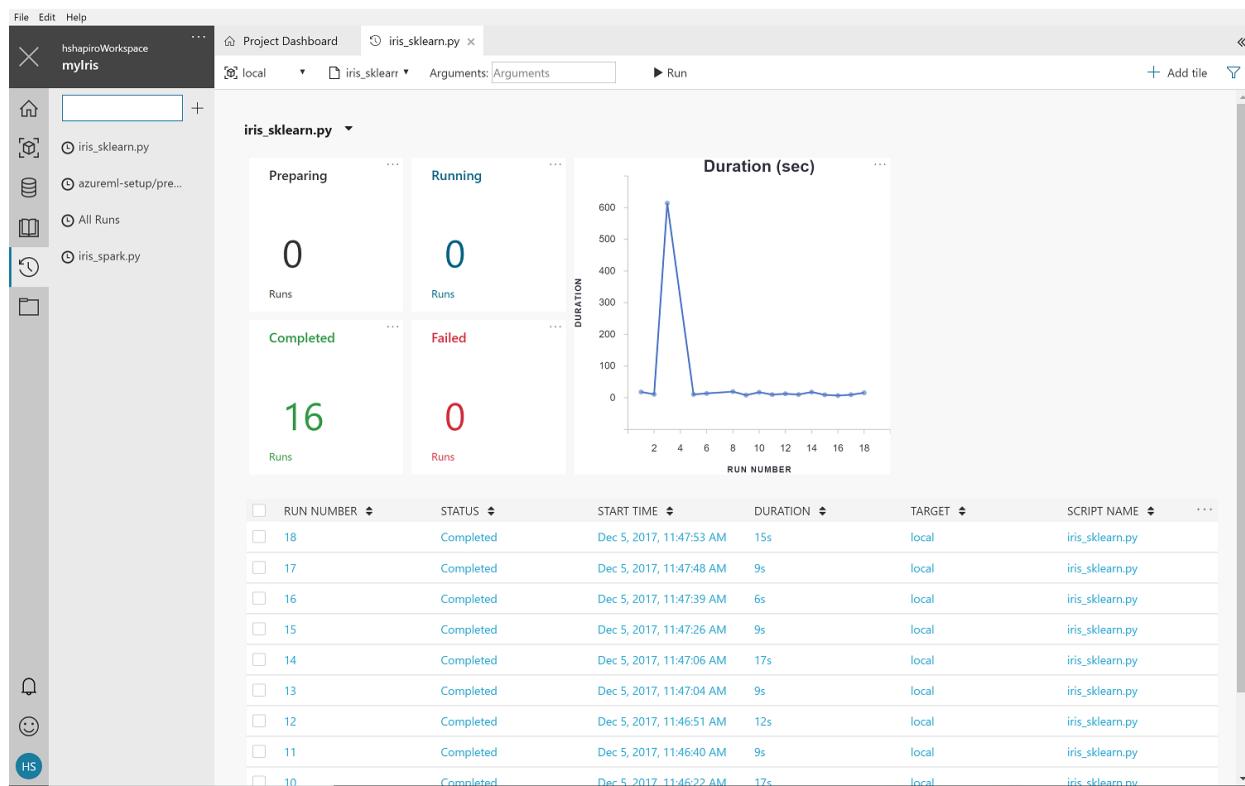
Many of the following updates are made as direct results of your feedback. Please keep them coming!

Notable New Features and Changes

- Support for running your scripts on remote Ubuntu VMs natively on your own environment in addition to remote-docker based execution.
- New environment experience in Workbench App allows you to create compute targets and run configurations in addition to our CLI-based experience.



- Customizable Run History reports



Detailed Updates

Following is a list of detailed updates in each component area of Azure Machine Learning in this sprint.

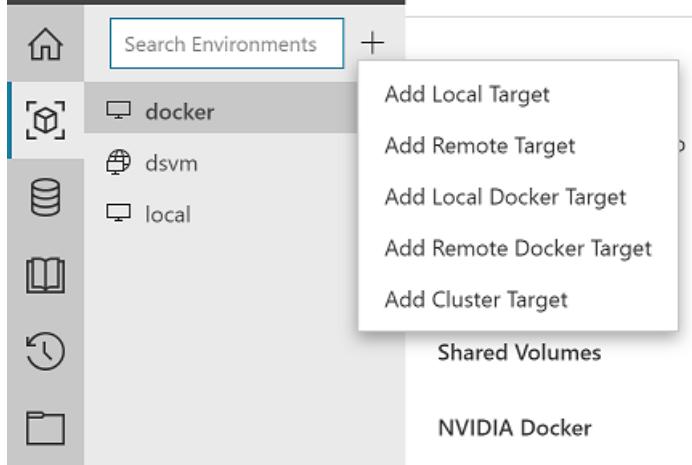
Workbench UI

- Customizable Run History reports
 - Improved chart configuration for Run History reports
 - The used entrypoints can be changed
 - Top-level filters can be added and modified

- Charts and stats can be added or modified (and drag-and-drop rearranged).

- CRUD for Run History reports
- Moved all existing run history list view configs to server-side reports, which acts like pipelines on runs from the selected entry points.
- Environments Tab

- Easily add new compute target and run configuration files to your project



- Manage and update your configuration files using a simple, form-based UX
- New button for preparing your environments for execution
- Performance improvements to the list of files in the sidebar

Data preparation

- Azure Machine Learning Workbench now allows you to be able to search for a column by using a known column's name.

Experimentation

- Azure Machine Learning Workbench now supports running your scripts natively on your own python or pyspark environment. For this capability, user creates and manages their own environment on the remote VM and use Azure Machine Learning Workbench to run their scripts on that target. Please see [Configuring Azure Machine Learning Experimentation Service](#)

Model Management

- Support for Customizing the Deployed Containers: enables customizing the container image by allowing installation of external libraries using apt-get, etc. It is no longer limited to pip-installable libraries. See the [documentation](#) for more info.

- Use the `--docker-file myDockerStepsFilename` flag and file name with the manifest, image, or service creation commands.
- Note that the base image is Ubuntu, and cannot be modified.
- Example command:

```
$ az ml image create -n myimage -m mymodel.pkl -f score.py --docker-file mydockerstepsfile
```

2018-01 (Sprint 3)

Version number: 0.1.1712.18263 ([Find your version](#))

The following are the updates and improvements in this sprint. Many of these updates are made as direct result of user feedback.

Following is a list of detailed updates in each component area of Azure Machine Learning in this sprint.

- Updates to the authentication stack forces login and account selection at startup

Workbench

- Ability to install/uninstall the app from Add/Remove Programs
- Updates to the authentication stack forces login and account selection at start-up
- Improved Single Sign On (SSO) experience on Windows
- Users that belong to multiple tenants with different credentials will now be able to sign into Workbench

UI

- General improvements and bug fixes

Notebooks

- General improvements and bug fixes

Data preparation

- Improved auto-suggestions while performing By Example transformations
- Improved algorithm for Pattern Frequency inspector
- Ability to send sample data and feedback while performing By Example transformations

BostonWeather								
DERIVE COLUMN BY EXAMPLE: You have selected 1 source column and provided 1 example. Review next suggested row Advanced mode Send feedback								
Columns: 8 Rows: 18943								
	abc DATE	abc DATE_1	abc DATE_2	abc Column (checkbox checked)	abc Column_1	abc HOURLYD...	abc HOURLYR...	abc HO
1	1/1/2015 0:54	1/1/2015	0:54	12AM-2AM	12-2	22	50	10
2	1/1/2015 1:54	1/1/2015	1:54	12AM-2AM	12-2	22	50	10
3	1/1/2015 2:54	1/1/2015	2:54	2AM-4AM	2-4	22	50	11
4	1/1/2015 3:54	1/1/2015	3:54	2AM-4AM	2-4	24	46	13
5	1/1/2015 4:54	1/1/2015	4:54	4AM-6AM	4-6	22	52	15
6	1/1/2015 5:54	1/1/2015	5:54	4AM-6AM	4-6	22	50	17

- Spark Runtime Improvements
- Scala has replaced Pyspark
- Fixed inability to close Data Not Applicable for the Time Series Inspector
- Fixed the hang time for Data Prep execution for HDI

Model Management CLI updates

- Ownership of the subscription is no longer required for provisioning resources. Contributor access to the resource group will be sufficient to set up the deployment environment.
- Enabled local environment setup for free subscriptions

2017-12 (Sprint 2 QFE)

Version number: 0.1.1711.15323 ([Find your version](#))

This is the QFE (Quick Fix Engineering) release, a minor release. It addresses several telemetry issues and helps the product team to better understand how the product is being used. The knowledge can go into future efforts for improving the product experience.

In addition, there are two important updates:

- Fixed a bug in data prep that prevented the time series inspector from displaying in data preparation packages.
- In the command-line tool, you no longer need to be an Azure subscription owner to provision Machine Learning Compute ACS clusters.

2017-12 (Sprint 2)

Version number: 0.1.1711.15263 ([Find your version](#))

Welcome to the third update of Azure Machine Learning. This update includes improvements in the workbench app, the Command-line Interface (CLI), and the back-end services. Thank you very much for sending the smiles

and frowns. Many of the following updates are made as direct results of your feedback.

Notable New Features

- Support for SQL Server and Azure SQL DB as a data source
- Deep Learning on Spark with GPU support using MMLSpark
- All AML containers are compatible with Azure IoT Edge devices when deployed (no extra steps required)
- Registered model list and detail views available Azure portal
- Accessing compute targets using SSH key-based authentication in addition to username/password-based access.
- New Pattern Frequency Inspector in the data prep experience.

Detailed Updates Following is a list of detailed updates in each component area of Azure Machine Learning in this sprint.

Installer

- Installer can self update so that bugs fixes and new features can be supported without user having to reinstall it

Workbench Authentication

- Multiple fixes to authentication system. Please let us know if you are still experiencing login issues.
- UI changes that make it easier to find the Proxy Manager settings.

Workbench

- Read-only file view now has light blue background
- Moved Edit button to the right to make it more discoverable.
- "dsource", "dprep", and "ipynb" file formats can now be rendered in raw text format
- The workbench now has a new editing experience that guides users towards using external IDEs to edit scripts, and use Workbench only to edit file types that have a rich editing experience (such as Notebooks, Data sources, Data preparation packages)
- Loading the list of workspaces and projects that the user has access to is significantly faster now

Data preparation

- A Pattern Frequency Inspector to view the string patterns in a column. You can also filter your data using these patterns. This shows you a view similar to the Value Counts inspector. The difference is that Pattern Frequency shows the counts of the unique patterns of the data, rather than the counts of unique data. You can also filter in or out all rows that fit a certain pattern.

18	329	Road End Caps	EC-R098	true
19	330	Touring End C...	EC-T209	true

INSPECTORS

Top 6 patterns of "ProductNumber"

Pattern	Percentage
[Upper]{2} & [Alpha Dash]+ & [Digit]+	~43%
[Upper]{2} & Const[-] & [Alpha Digit]+ & Const[-] & [Alpha Digit]+	~43%
Const[PA-] & [Digit]{3} & [Upper]{1}	~14%

- Performance improvements while recommending edge cases to review in the 'derive column by example' transformation
- Support for SQL Server and Azure SQL DB as a data source

Add Data Source X

1. Data Store

Select server
Select the AzureSQL/SQL Server you would like to use.

Server Address: .

2. Server

3. Sampling

Trust Server?

Authentication Type: Windows

Database To Connect To: AdventureWorks2014

Query

Preview

#	LocationID	Name	# CostRate	# Availability	ModifiedDa
1	1	Tool Crib	0	0	2008-04-30
2	2	Sheet Metal R...	0	0	2008-04-30
3	3	Paint Shop	0	0	2008-04-30
4	4	Paint Storage	0	0	2008-04-30
5	5	Metal Storage	0	0	2008-04-30
6	6	Miscellaneous ...	0	0	2008-04-30
7	7	Finished Good...	0	0	2008-04-30

Previous Next Finish

- Enabled "At a glance" view of row and column counts

The screenshot shows the Azure ML Workbench interface. At the top, there's a navigation bar with 'Help', 'Project Dashboard', and 'BostonWeather'. Below that is a sub-navigation bar with 'Metrics' and 'Prepare'. On the left, there's a sidebar with a refresh icon, a plus sign, and some project names: 'bway-tr...', 'Works2...', and 'weather'. The main area is a data grid titled 'BostonWeather'. It shows 'Columns: 15' and 'Rows: 424521'. The first four columns are highlighted in green. The data grid contains four rows of data:

	# tripduration	starttime	
bway-tr...	1	542	2015-01-01 00...
bway-tr...	2	438	2015-01-01 00...
Works2...	3	254	2015-01-01 00...
weather	4	432	2015-01-01 00...

- Data prep is enabled in all compute contexts
- Data sources that use a SQL Server database are enabled in all compute contexts
- Data prep grid columns can be filtered by data type
- Fixed issue with converting multiple columns to date
- Fixed issue that user could select output column as a source in Derive Column By Example if user changed output column name in the advanced mode.

Job execution

You can now create and access a remotedocker or cluster type compute target using SSH key-based authentication following these steps:

- Attach a compute target using the following command in CLI

```
$ az ml computetarget attach remotedocker --name "remotevm" --address "remotevm_IP_address" --username "sshuser" --use-azureml-ssh-key
```

NOTE

-k (or --use-azureml-ssh-key) option in the command specifies to generate and use SSH-key.

- Azure ML Workbench will generate a public key and output that in your console. Log into the compute target using the same username and append `~/.ssh/authorized_keys` file with this public key.
- You can prepare this compute target and use it for execution and Azure ML Workbench will use this key for authentication.

For more information on creating compute targets, see [Configuring Azure Machine Learning Experimentation Service](#)

Visual Studio Tools for AI

- Added support for [Visual Studio Tools for AI](#).

Command Line Interface (CLI)

- Added `az ml datasource create` command allows to creating a data source file from the command-line

Model Management and Operationalization

- [All AML containers are compatible with Azure IoT Edge devices when operationalized \(no extra steps required\)](#)
- Improvements of error messages in the o16n CLI
- Bug fixes in model management portal UX

- Consistent letter casing for model management attributes in the detail page
- Realtime scoring calls timeout set to 60 seconds
- Registered model list and detail views available in the Azure portal

Model "demo-iris-model.pkl"

Version	1
ID	4262b3caded0446390c4e705f1436a5e
Date created	9/15/2017, 5:38:36 PM
Location	https://modelmanagement.demoeus2e.blob.core.windows.net/
Description	Demo iris model
Tags	demo-iris-model

Services using "demo-iris-model.pkl"

Service	Last Updated	Image	Status
jesnewservice	9/28/2017, 6:51:02 PM	demo-iris-manifest	Succeeded
jesmonday	9/28/2017, 6:49:20 PM	demo-iris-manifest	Succeeded

Deploy and Manage your Machine Learning models.

Model
Register your models with the Model Management Account

Manifest
Specify registered models, code and dependencies to create a Manifest

Image
Use the Manifest to create a Docker™ image for web service

Service
Use the Docker™ image to create a Service

Learn More

Let's get started!

We have put together a Do-it-yourself walkthrough to show you how to deploy and manage your models.

Congratulations on creating your Model Management Account. To get you started, we have gone ahead and pre-registered from the Iris Project with your account. We have also created a Manifest for you using the scoring file and the registered model. You will need to create an environment before you proceed with creating an image or a service. Learn more about creating environments.

MMLSpark

- Deep Learning on Spark with [GPU support](#)
- Support for Resource Manager templates for easy resource deployment
- Support for the SparklyR ecosystem
- [AZTK integration](#)

Sample projects

- [Iris](#) and [MMLSpark](#) samples updated with the new Azure ML SDK version

Breaking changes

- Promoted the `--type` switch in `az ml computetarget attach` to a subcommand.

○ `az ml computetarget attach --type remotedocker` is now `az ml computetarget attach remotedocker`

- o `az ml computetarget attach --type cluster` is now `az ml computetarget attach cluster`

2017-11 (Sprint 1)

Version number: 0.1.1710.31013 ([Find your version](#))

In this release, we've made improvements around security, stability, and maintainability in the workbench app, the CLI, and the back-end services layer. Thanks very much for sending us smiles and frowns. Many of the below updates are made as direct results of your feedback. Keep them coming!

Notable New Features

- Azure ML is now available in two new Azure regions: **West Europe** and **Southeast Asia**. They join the previous regions of **East US 2**, **West Central US**, and **Australia East**, bringing the total number of deployed regions to five.
- We enabled Python code syntax highlighting in the Workbench app to make it easier to read and edit Python source code.
- You can now launch your favorite IDE directly from a file, rather than from the whole project. Opening a file in Workbench and then clicking "Edit" launches your IDE (currently VS Code and PyCharm are supported) to the current file and project. You can also click the arrow next to the Edit button to edit the file in the Workbench text editor. Files are read-only until you click Edit, preventing accidental changes.
- The popular plotting library `matplotlib` version 2.1.0 is now shipped with the Workbench app.
- We upgraded the .NET Core version to 2.0 for the data prep engine. This removed the requirement for brew-install openssl during app installation on macOS. It also paves the way for more exciting data prep features to come in the near future.
- We have enabled a version-specific app homepage, so you get more relevant release notes and update prompts based on your current app version.
- If your local user name has a space in it, the application can now be successfully installed.

Detailed Updates

Below is a list of detailed updates in each component area of Azure Machine Learning in this sprint.

Installer

- App installer now cleans up the install directory created by older version of the app.
- Fixed a bug that leads installer getting stuck at 100% on macOS High Sierra.
- There is now a direct link to the installer directory for user to review installer logs in case installation fails.
- Install now works for users that have space in their user name.

Workbench Authentication

- Support for authentication in Proxy Manager.
- Logging in now succeeds if user is behind a firewall.
- If user has experimentation accounts in multiple Azure regions, and if one region happens to be unavailable, the app no longer hangs.
- When authentication is not completed and the authentication dialog box is still visible, app no longer tries to load workspace from local cache.

Workbench App

- Python code syntax highlighting is enabled in text editor.
- The Edit button in the text editor allows you to edit the file either in an IDE (VS Code and PyCharm are supported) or in the built-in text editor.
- Text editor is in read-only mode by default.
- Save button visual state now changes to disabled after the current file is saved and hence no longer dirty.
- Workbench saves *all* unsaved files when you initiate a run.
- Workbench remembers the last used Workspace on the local machine so it opens automatically.
- Only a single instance of Workbench is now permitted to run. Previously multiple instances could be launched which caused issues when operating on the same project.

- Renamed File menu "Open Project..." to "Add Existing Folder as Project..."
- Tab switching is now a lot quicker.
- Help links are added to the Configuring IDE dialog box.
- The feedback form now remembers the email address you entered last time.
- Smiles and frowns form text area is now bigger, so you can send us more feedback!
- The `--owner` switch help text in `az ml workspace create` is corrected.
- We added an "About" dialog box to help user easily view and copy version number of the app.
- A "Suggest a feature" menu item is added to the Help menu.
- Experimentation account name is now visible in the app title bar, preceding the app name "Azure Machine Learning Workbench".
- A version-specific app homepage is displayed now based on the version of the app detected.

Data preparation

- External web site can no longer be loaded from Map Inspector to prevent potential security problems.
- Histogram and Value Count inspectors now has option to show graph in logarithmic scale.
- When a calculation is ongoing, data quality bar now shows a different color to signal the "calculating" state.
- Column metrics now show statistics for categorical value columns.
- The last character in the data source name is no longer truncated.
- Data prep package now remains open when switching tabs, resulting noticeable performance gains.
- In data source, when switching between data view and metric view, the order of columns now longer changes.
- Opening an invalid `.dprep` or `.dsource` file no longer causes Workbench to crash.
- Data prep package can now uses relative path for output in *Write to CSV* transform.
- Keep Column* transform now allows user to add additional columns when edited.
- Replace this* menu now actually launches *Replace Value* dialog box.
- Replace Value* transform now functions as expected instead of throwing error.
- Data prep package now uses absolute path when referencing data files outside of the project folder, making it possible to run the package in local context with absolute path to the data file.
- Full file* as a sampling strategy is now supported when using Azure blob as data source.
- Generated Python code (from data prep package) now carries both CR and LF, making it friendly in Windows.
- Choose Metrics* dropdown now hides property when switching to the Data view.
- Workbench can now process parquet files even when it is using Python runtime. Previously only Spark can be used when processing parquet files.
- Filtering out values in a column with *date* data type no longer causes data prep engine to crash.
- Metric view now respects sampling strategy updates.
- Remote sampling jobs now functions properly.

Job execution

- Argument is now included in run history record.
- Jobs kicked off in CLI now shows up in Run history Job panel automatically.
- Job panel now shows jobs created by guest users added to the Azure AD tenant.
- Job panel cancel and delete actions are more stable.
- When clicking on Run button, error message is triggered now if the configuration files are in bad format.
- Terminating app no longer interferes with jobs kicked off in CLI.
- Jobs kicked off in CLI now continues to spit out standard-out even after one hour of execution.
- Better error messages are shown when data prep package run fails in Python/PySpark.
- `az ml experiment clean` now cleans up Docker images in remote VM as well.
- `az ml experiment clean` now works properly for local target on macOS.
- Error messages when targeting local or remote Docker runs are cleaned up and easier to read.
- Better error message is displayed when HDInsight cluster head node name is not properly formatted when

attached as an execution target.

- Better error message is shown when secret is not found in the credential service.
- MMLSpark library is upgraded to support Apache Spark 2.2.
- MMLSpark now include subject encoding transform (Mesh encoding) for medical documents.
- `matplotlib` version 2.1.0 is now shipped out-of-the box with Workbench.

Jupyter Notebook

- Notebook name search now works properly in the Notebooks view.
- You can now delete a Notebook in the Notebooks view.
- New magic `%upload_artifact` is added for uploading files produced in the Notebook execution environment into run history data store.
- Kernel errors are now surfaced in Notebook job status for easier debugging.
- Jupyter server now properly shuts down when user logs out of the app.

Azure portal

- Experimentation account and Model Management account can now be created in two new Azure regions: West Europe and Southeast Asia.
- Model Management account DevTest plan now is only available when it is the first one to be created in the subscription.
- Help link in the Azure portal is updated to point to the correct documentation page.
- Description field is removed from Docker image details page since it is not applicable.
- Details including AppInsights and auto-scale settings are added to the web service detail page.
- Model management page now renders even if third-party cookies are disabled in the browser.

Operationalization

- Web service with "score" in its name no longer fails.
- User can now create a deployment environment with just Contributor access to an Azure resource group or the subscription. Owner access to the entire subscription is no longer needed.
- Operationalization CLI now enjoys tab auto-completion on Linux.
- Image construction service now supports building images for Azure IoT services/devices.

Sample projects

- [Classifying Iris](#) sample project:
 - `iris_pyspark.py` is renamed to `iris_spark.py`.
 - `iris_score.py` is renamed to `score_iris.py`.
 - `iris.dprep` and `iris.dsourc`e are updated to reflect the latest data prep engine updates.
 - `iris.ipynb` Notebook is amended to work in HDInsight cluster.
 - Run history is turned on in `iris.ipynb` Notebook cell.
- [Advanced Data Prep using Bike Share Data](#) sample project "Handle Error Value" step fixed.
- [MMLSpark on Adult Census Data](#) sample project `docker.runconfig` format updated from JSON to YAML.
- [Distributed Hyperparameter Tuning](#) sample project `docker.runconfig` format updated from JSON to YAML.
- New sample project [Image Classification using CNTK](#).

2017-10 (Sprint 0)

Version number: 0.1.1710.31013 ([Find your version](#))

Welcome to the first update of Azure Machine Learning Workbench following our initial public preview at the Microsoft Ignite 2017 conference. The main updates in this release are reliability and stabilization fixes. Some of the critical issues we addressed include:

New features

- macOS High Sierra is now supported

Bug fixes

Workbench experience

- Drag and drop a file into Workbench causes the Workbench to crash.
- The terminal window in VS Code configured as an IDE for Workbench does not recognize `az ml` commands.

Workbench authentication

We made a number of updates to improve various login and authentication issues reported.

- Authentication window keeps popping-up, particularly when Internet connection is not stable.
- Improved reliability issues around authentication token expiration.
- In some cases, authentication window appears twice.
- Workbench main window still displays "authenticating" message when the authentication process has finished and the pop-up dialog box already dismissed.
- If there is no Internet connection, the authentication dialog pops up with a blank screen.

Data preparation

- When a specific value is filtered, errors and missing values are also filtered out.
- Changing a sampling strategy removes subsequent existing join operations.
- Replacing Missing Value transform does not take NaN into consideration.
- Date type inference throws exception when null value encountered.

Job execution

- There is no clear error message when job execution fails to upload project folder because it exceeded the size limit.
- If user's Python script changes the working directory, the files written to outputs folders are not tracked.
- If the active Azure subscription is different than the one the current project belongs to, job submission results a 403 error.
- When Docker is not present, no clear error message is returned if user tries to use Docker as an execution target.
- `.runconfig` file is not saved automatically when user clicks on *Run* button.

Jupyter Notebook

- Notebook server cannot start if user uses with certain login types.
- Notebook server error messages do not surface in logs visible to user.

Azure portal

- Selecting the dark theme of Azure portal causes Model Management blade to display as a black box.

Operationalization

- Reusing a manifest to update a web service causes a new Docker image built with a random name.
- Web service logs cannot be retrieved from Kubernetes cluster.
- Misleading error message is printed when user attempts to create a Model Management account or an ML Compute account and encounters permissions issues.

Next steps

Read the overview for [Azure Machine Learning](#).

Quickstart: Use the Azure portal to get started with Azure Machine Learning

12/11/2018 • 4 minutes to read • [Edit Online](#)

In this quickstart, you use the Azure portal to create an Azure Machine Learning workspace. This workspace is the foundational block in the cloud that you use to experiment, train, and deploy machine learning models with Machine Learning. This quickstart uses cloud resources and requires no installation. To configure your own Jupyter notebook server instead, see [Quickstart: Use Python to get started with Azure Machine Learning](#).

In this quickstart, you:

- Create a workspace in your Azure subscription.
- Try it out with Python in an Azure notebook, and log values across multiple iterations.
- View the logged values in your workspace.

The following Azure resources are added automatically to your workspace when they're regionally available:

- [Azure Container Registry](#)
- [Azure Storage](#)
- [Azure Application Insights](#)
- [Azure Key Vault](#)

The resources you create can be used as prerequisites to other Machine Learning service tutorials and how-to articles. As with other Azure services, there are limits on certain resources associated with Machine Learning, such as compute cluster size. Learn more about [the default limits and how to increase your quota](#).

If you don't have an Azure subscription, create a free account before you begin. Try the [free or paid version of Azure Machine Learning service](#) today.

Create a workspace

Sign in to the [Azure portal](#) by using the credentials for the Azure subscription you use.

The portal's workspace dashboard is supported on Edge, Chrome, and Firefox browsers only.

All resources
ALL SUBSCRIPTIONS

No resources to display
Try changing your filters if you don't see what you're looking for.
[Learn more](#)

[Create resources](#)

Quickstarts + tutorials

[Windows Virtual Machines](#) Provision Windows Server, SQL Server, SharePoint VMs

[Linux Virtual Machines](#) Provision Ubuntu, Red Hat, CentOS, SUSE, CoreOS VMs

[App Service](#) Create Web Apps using .NET, Java, Node.js, Python, PHP

[Functions](#) Process events with a serverless code architecture

[SQL Database](#) Managed relational SQL Database as a Service

[Marketplace](#)

In the upper-left corner of the portal, select **Create a resource**.

[Create a resource](#)

All services

FAVORITES

In the search bar, enter **Machine Learning**. Select the **Machine Learning service workspace** search result.

Everything

Filter

machine learning service workspace

NAME	PUBLISHER	CATEGORY
Machine Learning service workspace	Microsoft	Analytics
Machine Learning Studio Workspace	Microsoft	Analytics
Machine Learning Studio Web Service Plan	Microsoft	Analytics

In the **Machine Learning service workspace** pane, scroll to the bottom and select **Create** to begin.

Create

In the **ML service workspace** pane, configure your workspace.

FIELD	DESCRIPTION
Workspace name	Enter a unique name that identifies your workspace. Here we use docs-ws. Names must be unique across the resource group. Use a name that's easy to recall and differentiate from workspaces created by others.
Subscription	Select the Azure subscription that you want to use.
Resource group	Use an existing resource group in your subscription, or enter a name to create a new resource group. A resource group is a container that holds related resources for an Azure solution. Here we use docs-aml.
Location	Select the location closest to your users and the data resources. This location is where the workspace is created.

ML service workspace □ X
Machine Learning service workspace

* Workspace name

* Subscription

* Resource group
 [Create new](#)

* Location

i For your convenience, these resources are added automatically to the workspace, if regionally available: [Azure Container Registry](#), [Azure storage](#), [Azure Application Insights](#) and [Azure Key Vault](#).

Create [Automation options](#)

To start the creation process, select **Create**. It can take a few moments to create the workspace.

To check on the status of the deployment, select the Notifications icon (bell) on the toolbar.



When the process is finished, a deployment success message appears. It's also present in the notifications section. To view the new workspace, select **Go to resource**.

On the workspace page, select [Explore your Azure Machine Learning service workspace](#).

Getting Started



Explore your Azure Machine Learning service workspace

Explore your Machine Learning service workspace to run and track experiments, compare model performance, and deploy models.

Use the workspace

Now see how a workspace helps you manage your machine learning scripts. In this section, you:

- Open a notebook in Azure Notebooks.
- Run code that creates some logged values.
- View the logged values in your workspace.

This example shows how the workspace can help you keep track of information generated in a script.

Open a notebook

Azure Notebooks provides a free cloud platform for Jupyter notebooks that are preconfigured with everything you need to run Machine Learning.

Select [Open Azure Notebooks](#) to try your first experiment.

The screenshot shows the Azure Machine Learning workspace interface. At the top, there's a header with the workspace name "docs-ws" and a "Machine Learning Workspace" label. Below the header is a navigation bar with tabs: Experiments (which is underlined), Pipelines, Compute, Models, Images, Deployments, and Activities. The main content area has a heading "Welcome to your new Workspace". Below it, there are two sections: "1. Getting started" and "2. Done getting started?". The "1. Getting started" section contains a button labeled "Open Azure Notebooks" which is highlighted with a red box. Below this button is a link "View More Sample Notebooks". The "2. Done getting started?" section contains a button labeled "View Experiments".

Your organization might require [administrator consent](#) before you can sign in.

After you sign in, a new tab opens and a [Clone Library](#) prompt appears. Select [Clone](#).

Run the notebook

Along with two notebooks, you see a [config.json](#) file. This config file contains information about the workspace you created.

Select [01.run-experiment.ipynb](#) to open the notebook.

To run the cells one at a time, use [Shift + Enter](#). Or select [Cells](#) > [Run All](#) to run the entire notebook.

When you see an asterisk [*] next to a cell, it's running. After the code for that cell finishes, a number appears.

After you've completed running all of the cells in the notebook, you can view the logged values in your workspace.

View logged values

After you run all the cells in the notebook, go back to the portal page.

Select `View Experiments`.

2. Done getting started?

Once you run the Azure Notebook, you will be able to view the data from the experiment in the Experiments page.

`View Experiments`

Close the `Reports` pop-up.

Select `my-first-experiment`.

See information about the run you just performed. Scroll down the page to find the table of runs. Select the run number link.

my-first-experiment

All Runs Report

[+ Add Chart](#) [Filter](#) [Get Link](#) [Refresh](#)

Filters

`parentRunId == null`

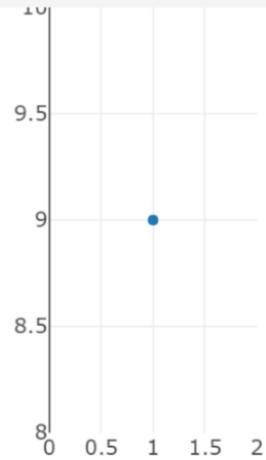
PREPARING

0

RUNNING

0

DURATION (SEC)



COMPLETED

1

FAILED

0

RUN NUMBER ↑↓

RUN ID ↑↓

STATUS ↑↓

CREATED TIME (UTC) ↑

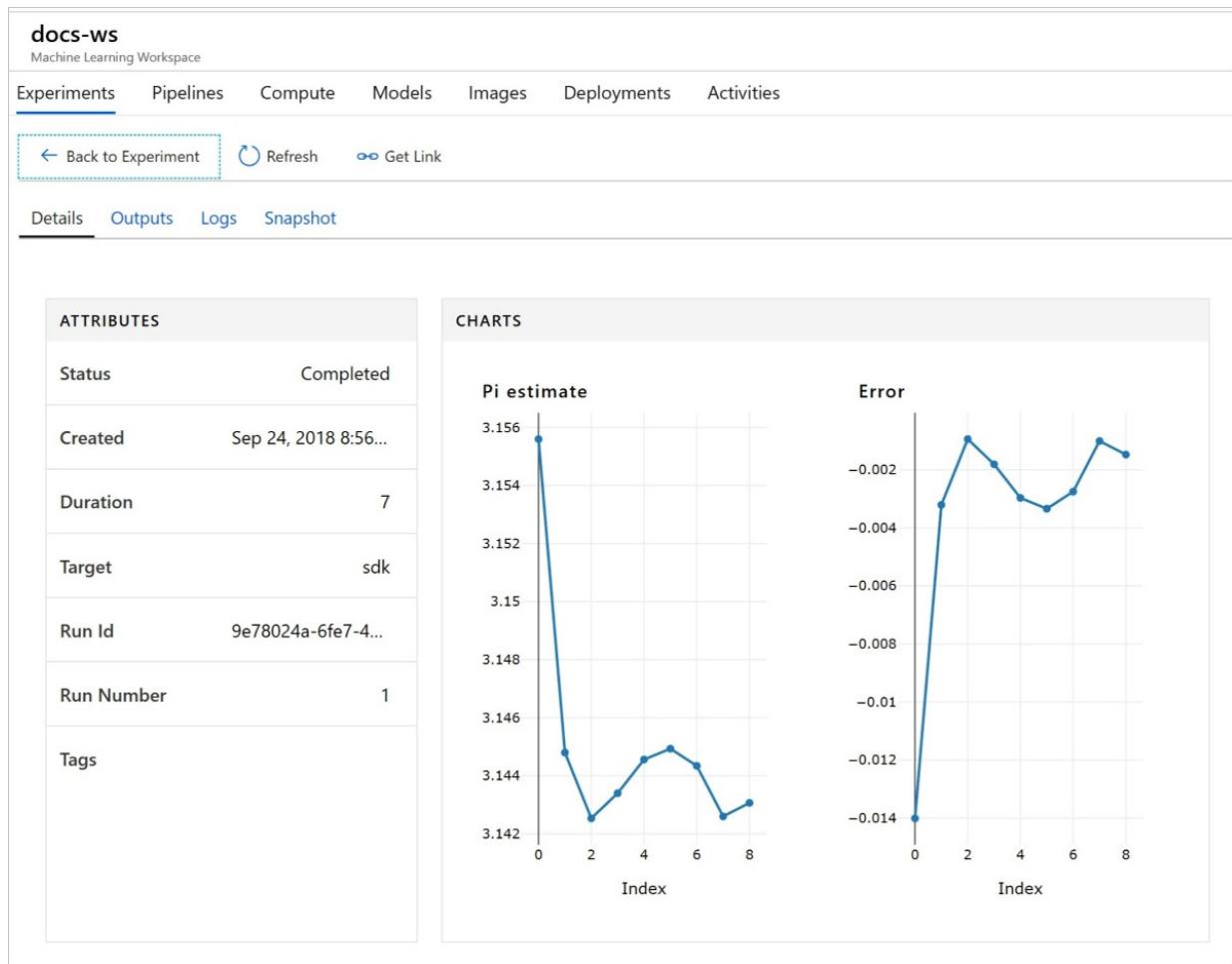
1

46800b79-6fb9-44b3-8e37-6f150bac7287

Completed

2018-09-07T16:32:32.849

You see plots that were automatically created of the logged values. Whenever you log multiple values with the same name parameter, a plot is automatically generated for you.



Since the code to approximate pi uses random values, your plots will show different values.

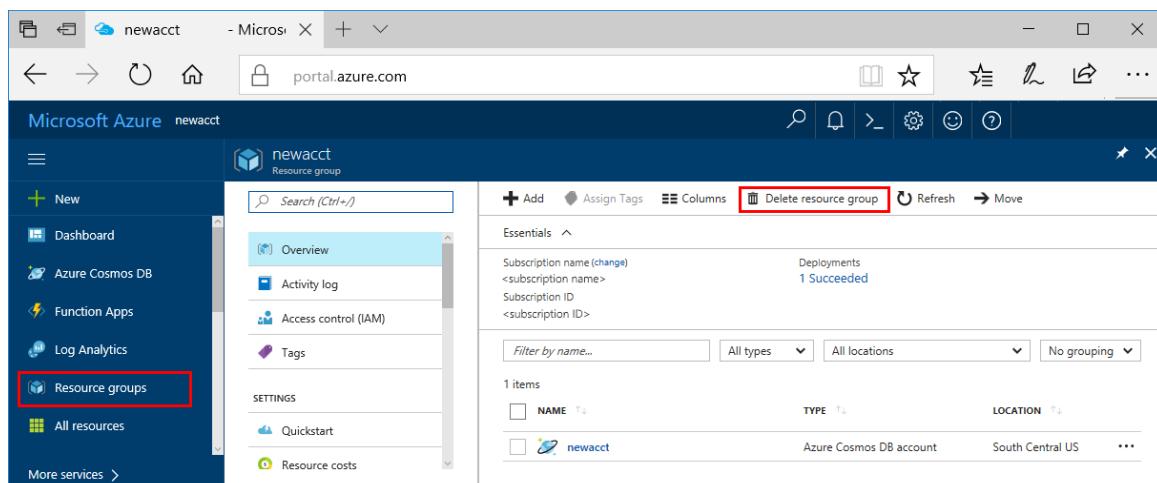
Clean up resources

IMPORTANT

The resources you created can be used as prerequisites to other Azure Machine Learning service tutorials and how-to articles.

If you don't plan to use the resources you created here, delete them so you don't incur any charges.

1. In the Azure portal, select **Resource groups** on the far left.



2. From the list, select the resource group you created.

3. Select **Delete resource group**.

4. Enter the resource group name, and then select **Delete**.

You also can keep the resource group but delete a single workspace. Display the workspace properties, and select **Delete**.

Next steps

You created the necessary resources to experiment with and deploy models. You also ran some code in a notebook. And you explored the run history from that code in your workspace in the cloud.

For an in-depth workflow experience, follow Machine Learning tutorials to train and deploy a model.

[Tutorial: Train an image classification model](#)

Quickstart: Use Python SDK to get started with Azure Machine Learning

12/11/2018 • 5 minutes to read • [Edit Online](#)

In this quickstart, you use the Azure Machine Learning SDK for Python to create and then use a Machine Learning service [workspace](#). This workspace is the foundational block in the cloud that you use to experiment, train, and deploy machine learning models with Machine Learning. In this quickstart, you start by configuring your own Python environment and Jupyter notebook server. To run with no installation, see [Quickstart: Use the Azure portal to get started with Azure Machine Learning](#).

In this tutorial, you install the Python SDK and:

- Create a workspace in your Azure subscription.
- Create a configuration file for that workspace to use later in other notebooks and scripts.
- Write code that logs values inside the workspace.
- View the logged values in your workspace.

In this quickstart, you create a workspace and a configuration file. You can use them as prerequisites to other Machine Learning tutorials and how-to articles. As with other Azure services, there are limits and quotas associated with Machine Learning. [Learn about quotas and how to request more](#).

The following Azure resources are added automatically to your workspace when they're regionally available:

- [Azure Container Registry](#)
- [Azure Storage](#)
- [Azure Application Insights](#)
- [Azure Key Vault](#)

If you don't have an Azure subscription, create a free account before you begin. Try the [free or paid version of Azure Machine Learning service](#) today.

Install the SDK

NOTE

Code in this article requires Azure Machine Learning SDK version 1.0.2 or later.

Skip this section if you use a data science virtual machine created after September 27, 2018. Those data science virtual machines come with the Python SDK preinstalled.

Before you install the SDK, we recommend that you create an isolated Python environment. While this quickstart uses [Miniconda](#), you also can use full [Anaconda](#) installed or [Python virtualenv](#).

Install Miniconda

[Download](#) and install Miniconda. Select the Python 3.7 version or later. Don't select the Python 2.x version.

Create an isolated Python environment

Open a command-line window. Then create a new conda environment named `myenv` with Python 3.6.

```
conda create -n myenv -y Python=3.6
```

Activate the environment.

```
conda activate myenv
```

Install the SDK

In the activated conda environment, install the SDK. This code installs the core components of the Machine Learning SDK. It also installs a Jupyter Notebook server in the conda environment. The installation takes a few minutes to finish, depending on the configuration of your machine.

```
# install Jupyter
conda install nb_conda

# install the base SDK and Jupyter Notebook
pip install azureml-sdk[notebooks]
```

You can also use different "extra" keywords to install additional components of the SDK.

```
# install the base SDK and auto ml components
pip install azureml-sdk[automl]

# install the base SDK and model explainability component
pip install azureml-sdk[explain]

# install the base SDK and experimental components
pip install azureml-sdk[contrib]
```

Use this install instead in a Databricks environment.

```
# install the base SDK and automl components in Azure Databricks environment
# read more at: https://github.com/Azure/MachineLearningNotebooks/tree/master/databricks
pip install azureml-sdk[databricks]
```

Create a workspace

To launch the Jupyter Notebook, enter this command.

```
jupyter notebook
```

In the browser window, create a new notebook by using the default `Python 3` kernel.

To display the SDK version, enter the following Python code in a notebook cell and execute it.

```
import azureml.core
print(azureml.core.VERSION)
```

Create a new Azure resource group and a new workspace.

Find a value for `<azure-subscription-id>` in the [subscriptions list in the Azure portal](#). Use any subscription in which your role is owner or contributor.

```
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
)
```

Executing the preceding code might trigger a new browser window for you to sign into your Azure account. After you sign in, the authentication token is cached locally.

To see the workspace details, such as associated storage, container registry, and key vault, enter the following code.

```
ws.get_details()
```

Write a configuration file

Save the details of your workspace in a configuration file into the current directory. This file is called 'aml_config\config.json'.

This workspace configuration file makes it easy to load this same workspace later. You can load it with other notebooks and scripts in the same directory or a subdirectory.

```
# Create the configuration file.
ws.write_config()

# Use this code to load the workspace from
# other scripts and notebooks in this directory.
# ws = Workspace.from_config()
```

The `write_config()` API call creates the configuration file in the current directory. The `config.json` file contains the following script.

```
{
  "subscription_id": "<azure-subscription-id>",
  "resource_group": "myresourcegroup",
  "workspace_name": "myworkspace"
}
```

Use the workspace

Write some code that uses the basic APIs of the SDK to track experiment runs.

```

from azureml.core import Experiment

# create a new experiment
exp = Experiment(workspace=ws, name='myexp')

# start a run
run = exp.start_logging()

# log a number
run.log('my magic number', 42)

# log a list (Fibonacci numbers)
run.log_list('my list', [1, 1, 2, 3, 5, 8, 13, 21, 34, 55])

# finish the run
run.complete()

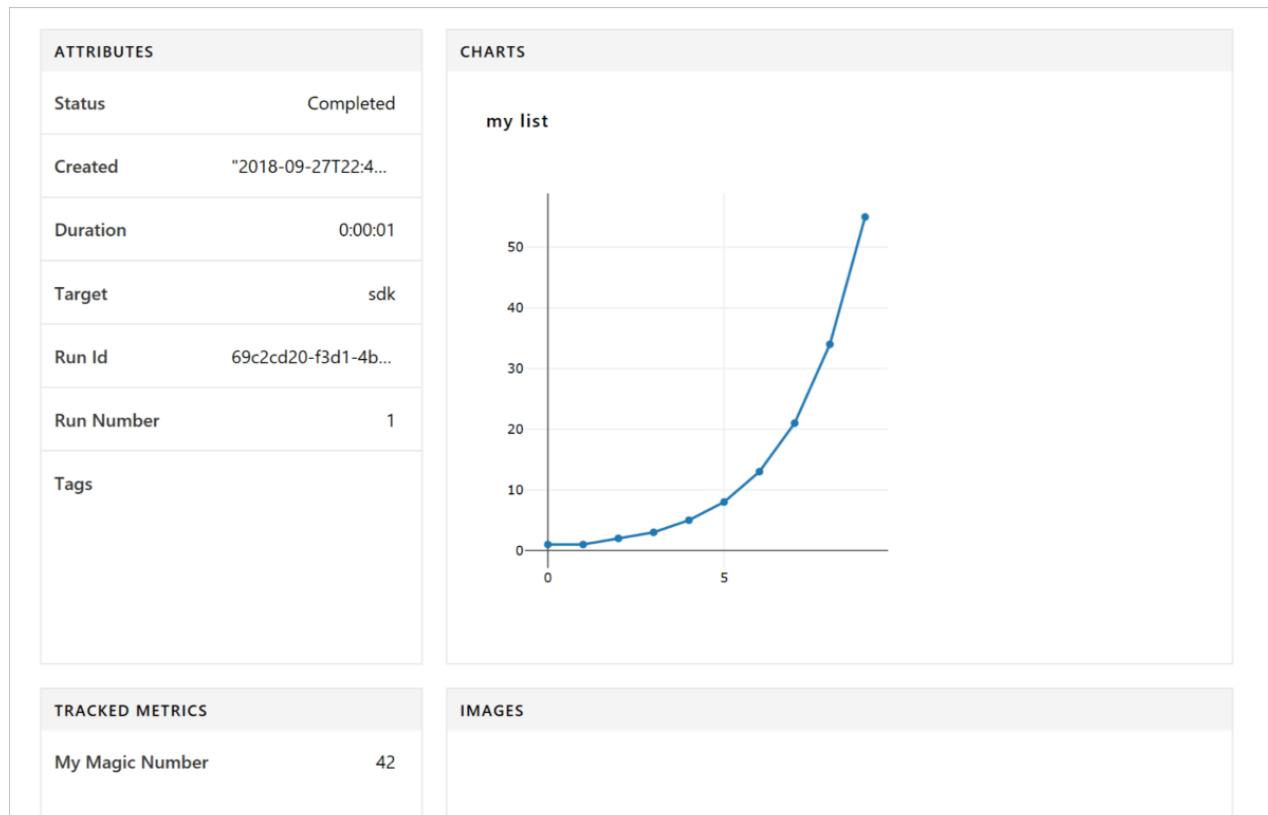
```

View logged results

When the run finishes, you can view the experiment run in the Azure portal. Use the following code to print a URL to the results for the last run.

```
print(run.get_portal_url())
```

Use the link to view the logged values in the Azure portal in your browser.



Clean up resources

IMPORTANT

The resources you created can be used as prerequisites to other Machine Learning tutorials and how-to articles.

If you don't plan to use the resources you created here, delete them so you don't incur any charges.

```
ws.delete(delete_dependent_resources=True)
```

Next steps

You created the necessary resources you need to experiment with and deploy models. You also ran code in a notebook. And you explored the run history from that code in your workspace in the cloud.

You need a few more packages in your environment to use it with Machine Learning tutorials.

1. In your browser, close your notebook.
2. In the command-line window, use **Ctrl + C** to stop the notebook server.
3. Install additional packages.

```
conda install -y cython matplotlib scikit-learn pandas numpy  
pip install azureml-sdk[automl]
```

After you install these packages, follow the tutorials to train and deploy a model.

[Tutorial: Train an image classification model](#)

You also can explore [more advanced examples on GitHub](#).

Tutorial: Train an image classification model with Azure Machine Learning service

12/12/2018 • 12 minutes to read • [Edit Online](#)

In this tutorial, you train a machine learning model both locally and on remote compute resources. You'll use the training and deployment workflow for Azure Machine Learning service in a Python Jupyter notebook. You can then use the notebook as a template to train your own machine learning model with your own data. This tutorial is **part one of a two-part tutorial series**.

This tutorial trains a simple logistic regression using the [MNIST](#) dataset and [scikit-learn](#) with Azure Machine Learning service. MNIST is a popular 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.

Learn how to:

- Set up your development environment
- Access and examine the data
- Train a simple logistic regression locally using the popular scikit-learn machine learning library
- Train multiple models on a remote cluster
- Review training results and register the best model

You'll learn how to select a model and deploy it in [part two of this tutorial](#) later.

If you don't have an Azure subscription, create a free account before you begin. Try the [free or paid version of Azure Machine Learning service](#) today.

NOTE

Code in this article was tested with Azure Machine Learning SDK version 1.0.2

Get the notebook

For your convenience, this tutorial is available as a [Jupyter notebook](#). Run the `tutorials/img-classification-part1-training.ipynb` notebook either in Azure Notebooks or in your own Jupyter notebook server.

Learn how to run notebooks by following the article, [Use Jupyter notebooks to explore this service](#).

Set up your development environment

All the setup for your development work can be accomplished in a Python notebook. Setup includes:

- Importing Python packages
- Connecting to a workspace to enable communication between your local computer and remote resources
- Creating an experiment to track all your runs
- Creating a remote compute target to use for training

Import packages

Import Python packages you need in this session. Also display the Azure Machine Learning SDK version.

```
%matplotlib inline
import numpy as np
import matplotlib
import matplotlib.pyplot as plt

import azureml
from azureml.core import Workspace, Run

# check core SDK version number
print("Azure ML SDK Version: ", azureml.core.VERSION)
```

Connect to workspace

Create a workspace object from the existing workspace. `Workspace.from_config()` reads the file **config.json** and loads the details into an object named `ws`.

```
# load workspace configuration from the config.json file in the current folder.
ws = Workspace.from_config()
print(ws.name, ws.location, ws.resource_group, ws.location, sep = '\t')
```

Create experiment

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

```
experiment_name = 'sklearn-mnist'

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

Create or attach existing AmlCompute

Azure Machine Learning Managed Compute(AmlCompute) is a managed service that enables data scientists to train machine learning models on clusters of Azure virtual machines, including VMs with GPU support. In this tutorial, you create AmlCompute as your training environment. This code creates the compute clusters for you if it does not already exist in your workspace.

Creation of the compute takes approximately 5 minutes. If the compute is already in the workspace this code uses it and skips the creation process.

```

from azureml.core.compute import AmlCompute
from azureml.core.compute import ComputeTarget
import os

# choose a name for your cluster
from azureml.core.compute import AmlCompute
from azureml.core.compute import ComputeTarget
import os

# choose a name for your cluster
compute_name = os.environ.get("AML_COMPUTE_CLUSTER_NAME", "cpucluster")
compute_min_nodes = os.environ.get("AML_COMPUTE_CLUSTER_MIN_NODES", 0)
compute_max_nodes = os.environ.get("AML_COMPUTE_CLUSTER_MAX_NODES", 4)

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

if compute_name in ws.compute_targets:
    compute_target = ws.compute_targets[compute_name]
    if compute_target and type(compute_target) is AmlCompute:
        print('found compute target. just use it. ' + compute_name)
    else:
        print('creating a new compute target...')
        provisioning_config = AmlCompute.provisioning_configuration(vm_size =
                                                                    min_nodes = compute_min_nodes,
                                                                    max_nodes = compute_max_nodes)

    # create the cluster
    compute_target = ComputeTarget.create(ws, compute_name, provisioning_config)

    # 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)

    # For a more detailed view of current AmlCompute status, use the 'status' property
    print(compute_target.status.serialize())

```

You now have the necessary packages and compute resources to train a model in the cloud.

Explore data

Before you train a model, you need to understand the data that you are using to train it. You also need to copy the data into the cloud so it can be accessed by your cloud training environment. In this section you learn how to:

- Download the MNIST dataset
- Display some sample images
- Upload data to the cloud

Download the MNIST dataset

Download the MNIST dataset and save the files into a `data` directory locally. Images and labels for both training and testing are downloaded.

```

import os
import urllib.request

os.makedirs('./data', exist_ok = True)

urllib.request.urlretrieve('http://yann.lecun.com/exdb/mnist/train-images-idx3-ubyte.gz',
filename='./data/train-images.gz')
urllib.request.urlretrieve('http://yann.lecun.com/exdb/mnist/train-labels-idx1-ubyte.gz',
filename='./data/train-labels.gz')
urllib.request.urlretrieve('http://yann.lecun.com/exdb/mnist/t10k-images-idx3-ubyte.gz',
filename='./data/test-images.gz')
urllib.request.urlretrieve('http://yann.lecun.com/exdb/mnist/t10k-labels-idx1-ubyte.gz',
filename='./data/test-labels.gz')

```

Display some sample images

Load the compressed files into `numpy` arrays. Then use `matplotlib` to plot 30 random images from the dataset with their labels above them. Note this step requires a `load_data` function that's included in an `util.py` file. This file is included in the sample folder. Please make sure it is placed in the same folder as this notebook. The `load_data` function simply parses the compressed files into numpy arrays.

```

# make sure utils.py is in the same directory as this code
from utils import load_data

# note we also shrink the intensity values (X) from 0-255 to 0-1. This helps 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 let's show some randomly chosen images from the training set.
count = 0
sample_size = 30
plt.figure(figsize = (16, 6))
for i in np.random.permutation(X_train.shape[0])[:sample_size]:
    count = count + 1
    plt.subplot(1, sample_size, count)
    plt.axhline('')
    plt.axvline('')
    plt.text(x=10, y=-10, s=y_train[i], fontsize=18)
    plt.imshow(X_train[i].reshape(28, 28), cmap=plt.cm.Greys)
plt.show()

```

A random sample of images displays:

8	9	4	7	9	2	9	4	7	8	4	8	8	2	3	2	4	6	1	1	3	8	8	1	3	1	8	4	1	8
8	9	4	7	9	2	9	4	7	8	4	8	8	2	3	2	4	6	1	1	3	8	8	1	3	1	8	4	1	8

Now you have an idea of what these images look like and the expected prediction outcome.

Upload data to the cloud

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 datastore is a convenient construct associated with your workspace for you to upload/download data, and interact with it from your remote compute targets. It is backed by Azure blob storage account.

The MNIST 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)
```

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

Train a local model

Train a simple logistic regression model using scikit-learn locally.

Training locally can take a minute or two depending on your computer configuration.

```
%%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))
```

The local model accuracy displays:

```
0.9202
```

With just a few lines of code, you have a 92% accuracy.

Train on a remote cluster

Now you can expand on this simple model by building a model with a different regularization rate. This time you'll train the model on a remote resource.

For this task, submit the job to the remote training cluster you set up earlier. To submit a job you:

- Create a directory
- Create a training script
- Create an estimator object
- Submit the job

Create a directory

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

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

Create a training script

To submit the job to the cluster, first create a training script. Run the following code to create the training script called `train.py` in the directory you just created. This training adds a regularization rate to the training algorithm, so produces a slightly different model than the local version.

```

%%writefile $script_folder/train.py

import argparse
import os
import numpy as np

from sklearn.linear_model import LogisticRegression
from sklearn.externals import joblib

from azureml.core import Run
from utils import load_data

# let user feed in 2 parameters, the location of the data files (from datastore), and the regularization
# rate of the logistic regression model
parser = argparse.ArgumentParser()
parser.add_argument('--data-folder', type=str, dest='data_folder', help='data folder mounting point')
parser.add_argument('--regularization', type=float, dest='reg', default=0.01, help='regularization rate')
args = parser.parse_args()

data_folder = os.path.join(args.data_folder, 'mnist')
print('Data folder:', data_folder)

# 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.
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()

print('Train a logistic regression model with regularization rate of', args.reg)
clf = LogisticRegression(C=1.0/args.reg, random_state=42)
clf.fit(X_train, y_train)

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)
# note file saved in the outputs folder is automatically uploaded into experiment record
joblib.dump(value=clf, filename='outputs/sklearn_mnist_model.pkl')

```

Notice how the script gets data and saves models:

- The training script reads an argument to find the directory containing the data. When you submit the job later, you point to the datastore for this argument:

```
parser.add_argument('--data-folder', type=str, dest='data_folder', help='data directory mounting point')
```

- The training script saves your model into a directory named outputs.

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

Anything written in this directory is automatically uploaded into your workspace. You'll access your model from this directory later in the tutorial. The file `utils.py` is referenced from the training script to load the dataset correctly. Copy this script into the script folder so that it can be accessed along with the

training script on the remote resource.

```
import shutil  
shutil.copy('utils.py', script_folder)
```

Create an estimator

An estimator object is used to submit the run. Create your estimator by running the following code to define:

- The name of the estimator object, `est`
- The directory that contains your scripts. All the files in this directory are uploaded into the cluster nodes for execution.
- The compute target. In this case you will use the Azure Machine Learning compute cluster you created
- The training script name, `train.py`
- Parameters required from the training script
- Python packages needed for training

In this tutorial, this target is `AmlCompute`. All files in the script folder are uploaded into the cluster nodes for execution. The `data_folder` is set to use the datastore (`ds.as_mount()`).

```
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'])
```

Submit the job to the cluster

Run the experiment by submitting the estimator object.

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

Since the call is asynchronous, it returns a **Preparing** or **Running** state as soon as the job is started.

Monitor a remote run

In total, the first run takes **approximately 10 minutes**. But for subsequent runs, as long as the script dependencies don't change, the same image is reused and hence the container start up time is much faster.

Here is what's happening while you wait:

- **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 stage 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.

You can check the progress of a running job in multiple ways. This tutorial uses a Jupyter widget as well as a `wait_for_completion` method.

Jupyter widget

Watch the progress of the run with a Jupyter 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()
```

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

Run Properties		Output Logs
Status	Completed	Uploading experiment status to history service. Adding run profile attachment azureml-logs/80_driver_log.txt
Start Time	8/10/2018 12:11:42 PM	Data folder: /mnt/batch/tasks/shared/LS_root/jobs/gpucluster225c81517743bf5/azureml/sklearn-mnist_1533921100384/mounts/workspacefilestore/mnist (60000, 784) (60000,) (10000, 784) (10000,)
Duration	0:07:20	Train a logistic regression model with regularization rate of 0.01
Run Id	sklearn-mnist_1533921100384	Predict the test set
Arguments	N/A	Accuracy is 0.9185
regularization rate	0.01	The experiment completed successfully. Starting post-processing steps.
accuracy	0.9185	

[Click here to see the run in Azure portal](#)

Get log results upon completion

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.

```
run.wait_for_completion(show_output=False) # specify True for a verbose log
```

Display run results

You now have a model trained on a remote cluster. Retrieve the accuracy of the model:

```
print(run.get_metrics())
```

The output shows the remote model has an accuracy slightly higher than the local model, due to the addition of the regularization rate during training.

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

In the next tutorial you will explore this model in more detail.

Register model

The last step in the training script 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.

You can see files associated with that run.

```
print(run.get_file_names())
```

Register the model in the workspace so that you (or other collaborators) can later query, examine, and deploy this model.

```
# register model
model = run.register_model(model_name='sklearn_mnist', model_path='outputs/sklearn_mnist_model.pkl')
print(model.name, model.id, model.version, sep = '\t')
```

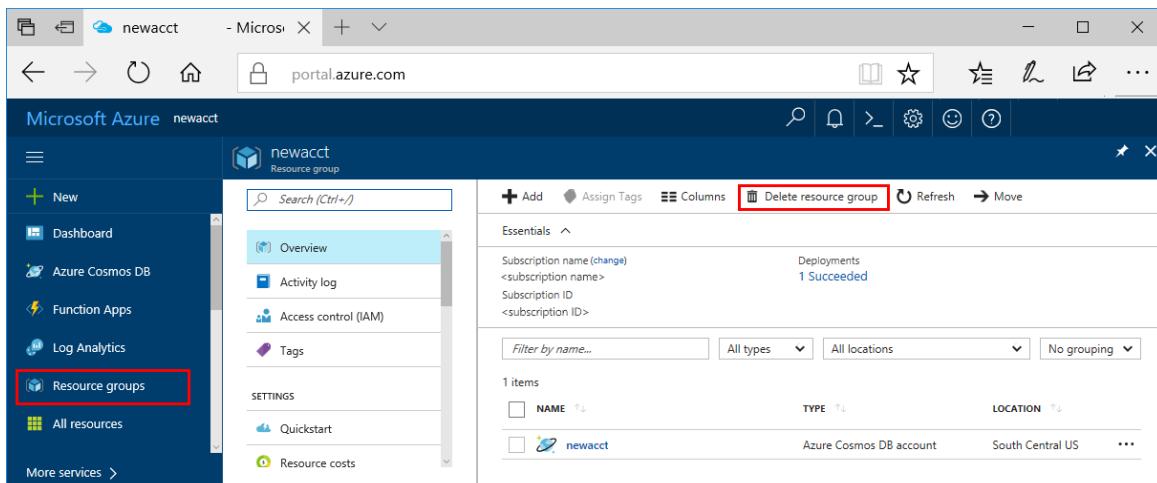
Clean up resources

IMPORTANT

The resources you created can be used as prerequisites to other Azure Machine Learning service tutorials and how-to articles.

If you don't plan to use the resources you created here, delete them so you don't incur any charges.

1. In the Azure portal, select **Resource groups** on the far left.



The screenshot shows the Microsoft Azure portal interface. The left sidebar has a 'Resource groups' link highlighted with a red box. The main content area shows the 'newacct' resource group details. At the top right of the main area, there is a 'Delete resource group' button, which is also highlighted with a red box.

2. From the list, select the resource group you created.
3. Select **Delete resource group**.
4. Enter the resource group name, and then select **Delete**.

You can also delete just the Azure Managed Compute cluster. However, since autoscale is turned on and the cluster minimum is 0, this particular resource will not incur additional compute charges when not in use.

```
# optionally, delete the Azure Managed Compute cluster
compute_target.delete()
```

Next steps

In this Azure Machine Learning service tutorial, you used Python to:

- Set up your development environment
- Access and examine the data
- Train a simple logistic regression locally using the popular scikit-learn machine learning library
- Train multiple models on a remote cluster
- Review training details and register the best model

You are ready to deploy this registered model using the instructions in the next part of the tutorial series:

[Tutorial 2 - Deploy models](#)

Tutorial: Deploy an image classification model in Azure Container Instance

12/12/2018 • 8 minutes to read • [Edit Online](#)

This tutorial is **part two of a two-part tutorial series**. In the [previous tutorial](#), you trained machine learning models and then registered a model in your workspace on the cloud.

Now, you're ready to deploy the model as a web service in [Azure Container Instances](#). A web service is an image, in this case a Docker image, that encapsulates the scoring logic and the model itself.

In this part of the tutorial, you use the Azure Machine Learning service to:

- Set up your testing environment
- Retrieve the model from your workspace
- Test the model locally
- Deploy the model to Container Instances
- Test the deployed model

Container Instances isn't ideal for production deployments, but it's great for testing and understanding the workflow. For scalable production deployments, consider using Azure Kubernetes Service. For more information, see [How to deploy and where](#).

Get the notebook

For your convenience, this tutorial is available as a [Jupyter notebook](#). Run the `tutorials/img-classification-part2-deploy.ipynb` notebook, either in Azure Notebooks or in your own Jupyter notebook server.

Learn how to run notebooks by following the article, [Use Jupyter notebooks to explore this service](#).

NOTE

Code in this article was tested with Azure Machine Learning SDK version 1.0.2.

Prerequisites

Complete the model training in the following notebook: [Tutorial #1: Train an image classification model with Azure Machine Learning service](#).

Set up the environment

Start by setting up a testing environment.

Import packages

Import the Python packages needed for this tutorial.

```
%matplotlib inline
import numpy as np
import matplotlib
import matplotlib.pyplot as plt

import azureml
from azureml.core import Workspace, Run

# display the core SDK version number
print("Azure ML SDK Version: ", azureml.core.VERSION)
```

Retrieve the model

You registered a model in your workspace in the previous tutorial. Now, load this workspace and download the model to your local directory.

```
from azureml.core import Workspace
from azureml.core.model import Model

ws = Workspace.from_config()
model=Model(ws, 'sklearn_mnist')
model.download(target_dir = '.')
import os
# verify the downloaded model file
os.stat('./sklearn_mnist_model.pkl')
```

Test model locally

Before deploying, make sure your model is working locally by:

- Loading test data
- Predicting test data
- Examining the confusion matrix

Load test data

Load the test data from the **./data/** directory created during the training tutorial.

```
from utils import load_data

# note we also shrink the intensity values (X) from 0-255 to 0-1. This helps the neural network converge
# faster

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

Predict test data

Feed the test dataset to the model to get predictions.

```
import pickle
from sklearn.externals import joblib

clf = joblib.load('./sklearn_mnist_model.pkl')
y_hat = clf.predict(X_test)
```

Examine the confusion matrix

Generate a confusion matrix to see how many samples from the test set are classified correctly. Notice the misclassified value for the incorrect predictions.

```

from sklearn.metrics import confusion_matrix

conf_mx = confusion_matrix(y_test, y_hat)
print(conf_mx)
print('Overall accuracy:', np.average(y_hat == y_test))

```

The output shows the confusion matrix:

```

[[ 960     0     1     2     1     5     6     3     1     1]
 [    0 1112     3     1     0     1     5     1    12     0]
 [    9     8 920    20    10     4    10    11    37     3]
 [    4     0    17 921     2    21     4    12    20     9]
 [    1     2     5     3 915     0     10     2     6    38]
 [   10     2     0    41    10   770    17     7    28     7]
 [    9     3     7     2     6    20   907     1     3     0]
 [    2     7    22     5     8     1     1 950     5    27]
 [   10    15     5    21    15    27     7    11 851    12]
 [    7     8     2    13    32    13     0    24    12 898]]

```

Overall accuracy: 0.9204

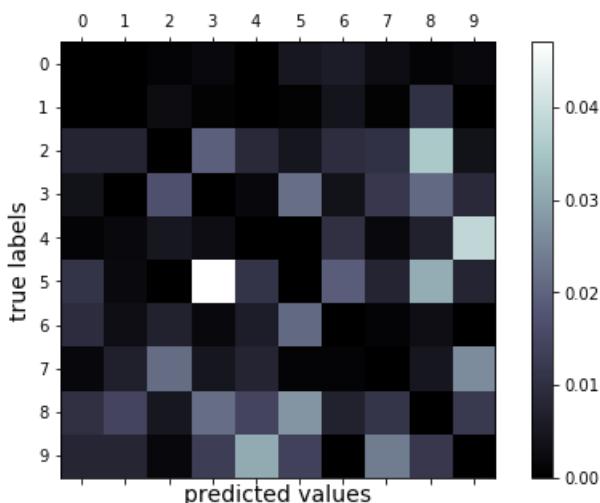
Use `matplotlib` to display the confusion matrix as a graph. In this graph, the X axis represents the actual values, and the Y axis represents the predicted values. The color in each grid represents the error rate. The lighter the color, the higher the error rate is. For example, many 5's are misclassified as 3's. Hence, you see a bright grid at (5,3).

```

# normalize the diagonal cells so that they don't overpower the rest of the cells when visualized
row_sums = conf_mx.sum(axis=1, keepdims=True)
norm_conf_mx = conf_mx / row_sums
np.fill_diagonal(norm_conf_mx, 0)

fig = plt.figure(figsize=(8,5))
ax = fig.add_subplot(111)
cax = ax.matshow(norm_conf_mx, cmap=plt.cm.bone)
ticks = np.arange(0, 10, 1)
ax.set_xticks(ticks)
ax.set_yticks(ticks)
ax.set_xticklabels(ticks)
ax.set_yticklabels(ticks)
fig.colorbar(cax)
plt.ylabel('true labels', fontsize=14)
plt.xlabel('predicted values', fontsize=14)
plt.savefig('conf.png')
plt.show()

```



Deploy as web service

Once you've tested the model and are satisfied with the results, deploy the model as a web service hosted in Container Instances.

To build the correct environment for Container Instances, provide the following:

- A scoring script to show how to use the model
- An environment file to show what packages need to be installed
- A configuration file to build the container instance
- The model you trained before

Create scoring script

Create the scoring script, called score.py. The web service call uses this to show how to use the model.

You must include two required functions into the scoring script:

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

```
%%writefile score.py
import json
import numpy as np
import os
import pickle
from sklearn.externals import joblib
from sklearn.linear_model import LogisticRegression

from azureml.core.model import Model

def init():
    global model
    # retrieve 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())
```

Create environment file

Next, 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 model 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())
```

Review the content of the `myenv.yml` file.

```
with open("myenv.yml","r") as f:  
    print(f.read())
```

Create configuration file

Create a deployment configuration file, and specify the number of CPUs and gigabyte of RAM needed for your Container Instances container. While it depends on your model, the default of 1 core and 1 gigabyte of RAM is usually sufficient for many models. If you feel you need more later, you would have to recreate the image and redeploy the service.

```
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')
```

Deploy in Container Instances

Estimated time to complete: **about 7-8 minutes**

Configure the image and deploy. The following code goes through these steps:

1. Build an image by using:
 - The scoring file (`score.py`).
 - The environment file (`myenv.yml`).
 - The model file.
2. Register that image under the workspace.
3. Send the image to the Container Instances container.
4. Start up a container in Container Instances by using the image.
5. Get the web service HTTP endpoint.

```
%%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)
```

Get the scoring web service's HTTP endpoint, which accepts REST client calls. You can share this endpoint with anyone who wants to test the web service or integrate it into an application.

```
print(service.scoring_uri)
```

Test deployed service

Earlier, you scored all the test data with the local version of the model. Now, you can test the deployed model with a random sample of 30 images from the test data.

The following code goes through these steps:

1. Send the data as a JSON array to the web service hosted in Container Instances.
2. Use the SDK's `run` API to invoke the service. You can also make raw calls by using any HTTP tool, such as curl.
3. Print the returned predictions, and plot them along with the input images. Red font and inverse image (white on black) is used to highlight the misclassified samples.

Since the model accuracy is high, you might have to run the following code a few times before you can see a misclassified sample.

```
import json

# find 30 random samples from test set
n = 30
sample_indices = np.random.permutation(X_test.shape[0])[0:n]

test_samples = json.dumps({"data": X_test[sample_indices].tolist()})
test_samples = bytes(test_samples, encoding = 'utf8')

# predict using the deployed model
result = json.loads(service.run(input_data=test_samples))

# compare actual value vs. the predicted values:
i = 0
plt.figure(figsize = (20, 1))

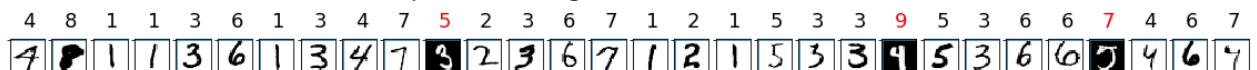
for s in sample_indices:
    plt.subplot(1, n, i + 1)
    plt.axhline('')
    plt.axvline('')

    # use different color for misclassified sample
    font_color = 'red' if y_test[s] != result[i] else 'black'
    clr_map = plt.cm.gray if y_test[s] != result[i] else plt.cm.Greys

    plt.text(x=10, y = -10, s=result[i], fontsize=18, color=font_color)
    plt.imshow(X_test[s].reshape(28, 28), cmap=clr_map)

    i = i + 1
plt.show()
```

Here is the result of one random sample of test images:

4 8 1 1 3 6 1 3 4 7 5 2 3 6 7 1 2 1 5 3 3 9 5 3 6 6 7 4 6 7


You can also send raw HTTP request to test the web service.

```

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'}

# for AKS deployment you'd need to add the service key in the header as well
# api_key = service.get_key()
# headers = {'Content-Type':'application/json', 'Authorization':('Bearer ' + api_key)}


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)

```

Clean up resources

To keep the resource group and workspace for other tutorials and exploration, you can delete only the Container Instances deployment by using this API call:

```
service.delete()
```

IMPORTANT

The resources you created can be used as prerequisites to other Azure Machine Learning service tutorials and how-to articles.

If you don't plan to use the resources you created here, delete them so you don't incur any charges.

1. In the Azure portal, select **Resource groups** on the far left.

The screenshot shows the Azure portal interface with the URL portal.azure.com. On the left, the navigation menu is open, and the 'Resource groups' option is highlighted with a red box. The main content area displays the 'newacct' resource group details. At the top right of the blade, there is a 'Delete resource group' button, which is also highlighted with a red box. The 'Essentials' section shows the subscription name as 'newacct' and the deployment status as '1 Succeeded'. Below that, a table lists one item: 'newacct' (Azure Cosmos DB account) located in South Central US.

NAME	TYPE	LOCATION
newacct	Azure Cosmos DB account	South Central US

2. From the list, select the resource group you created.
3. Select **Delete resource group**.
4. Enter the resource group name, and then select **Delete**.

Next steps

- Learn about all of the [deployment options for Azure Machine Learning service](#), including ACI, Azure Kubernetes Service, FPGAs, and IoT Edge.
- See how Azure Machine Learning service can auto-select and tune the best algorithm for your model, and build that model for you. Try out the [Automatic algorithm selection](#) tutorial.

Tutorial: Prepare data for regression modeling

12/12/2018 • 10 minutes to read • [Edit Online](#)

In this tutorial, you learn how to prep data for regression modeling using the Azure Machine Learning Data Prep SDK. Perform various transformations to filter and combine two different NYC Taxi data sets. The end goal of this tutorial set is to predict the cost of a taxi trip by training a model on data features including pickup hour, day of week, number of passengers, and coordinates. This tutorial is part one of a two-part tutorial series.

In this tutorial, you:

- Setup a Python environment and import packages
- Load two datasets with different field names
- Cleanse data to remove anomalies
- Transform data using intelligent transforms to create new features
- Save your dataflow object to use in a regression model

You can prepare your data in Python using the [Azure Machine Learning Data Prep SDK](#).

Get the notebook

For your convenience, this tutorial is available as a [Jupyter notebook](#). Run the `regression-part1-data-prep.ipynb` notebook either in Azure Notebooks or in your own Jupyter notebook server.

Learn how to run notebooks by following the article, [Use Jupyter notebooks to explore this service](#).

Import packages

Begin by importing the SDK.

```
import azureml.dataprep as dprep
```

Load data

Download two different NYC Taxi data sets into dataflow objects. These datasets contain slightly different fields.

The method `auto_read_file()` automatically recognizes the input file type.

```
dataset_root = "https://dprepdata.blob.core.windows.net/demo"

green_path = "/".join([dataset_root, "green-small/*"])
yellow_path = "/".join([dataset_root, "yellow-small/*"])

green_df = dprep.read_csv(path=green_path, header=dprep.PromoteHeadersMode.GROUPED)
# auto_read_file will automatically identify and parse the file type, and is useful if you don't know the file
# type
yellow_df = dprep.auto_read_file(path=yellow_path)

display(green_df.head(5))
display(yellow_df.head(5))
```

Cleanse data

Now you populate some variables with shortcut transforms that will apply to all dataflows. The variable `drop_if_all_null` will be used to delete records where all fields are null. The variable `useful_columns` holds an array of column descriptions that are retained in each dataflow.

```
all_columns = dprep.ColumnSelector(term=".*", use_regex=True)
drop_if_all_null = [all_columns, dprep.ColumnRelationship(dprep.ColumnRelationship.ALL)]
useful_columns = [
    "cost", "distance", "dropoff_datetime", "dropoff_latitude", "dropoff_longitude",
    "passengers", "pickup_datetime", "pickup_latitude", "pickup_longitude", "store_forward", "vendor"
]
```

You first work with the green taxi data and get it into a valid shape that can be combined with the yellow taxi data. Create a temporary dataflow `tmp_df`. Call the `replace_na()`, `drop_nulls()`, and `keep_columns()` functions using the shortcut transform variables you created. Additionally, rename all the columns in the dataframe to match the names in `useful_columns`.

```
tmp_df = (green_df
    .replace_na(columns=all_columns)
    .drop_nulls(*drop_if_all_null)
    .rename_columns(column_pairs={
        "VendorID": "vendor",
        "lpep_pickup_datetime": "pickup_datetime",
        "lpep_dropoff_datetime": "dropoff_datetime",
        "lpep_dropoff_datetime": "dropoff_datetime",
        "Store_and_fwd_flag": "store_forward",
        "store_and_fwd_flag": "store_forward",
        "Pickup_longitude": "pickup_longitude",
        "Pickup_latitude": "pickup_latitude",
        "Dropoff_longitude": "dropoff_longitude",
        "Dropoff_latitude": "dropoff_latitude",
        "Passenger_count": "passengers",
        "Fare_amount": "cost",
        "Trip_distance": "distance"
    })
    .keep_columns(columns=useful_columns))
tmp_df.head(5)
```

	VENDOR	PICKUP_DATETIME	DROPOFF_DATETIME	STORE_FORWARD	PICKUP_LONGITUDE	PICKUP_LATITUDE	DROPOFF_LONGITUDE	DROPOFF_LATITUDE	PASSENGERS	DISTANCE	COST
0	2	2013-08-01 08:14:37	2013-08-01 09:09:06	N	0	0	0	0	1	.00	21.25
1	2	2013-08-01 09:13:00	2013-08-01 11:38:00	N	0	0	0	0	2	.00	74.5
2	2	2013-08-01 09:48:00	2013-08-01 09:49:00	N	0	0	0	0	1	.00	1

	VENDOR	PICKUP_DATETIME	DROP_OFF_DATETIME	STORE_FORWARD	PICKUP_LONGITUDE	PICKUP_LATITUDE	DROP_OFF_LONGITUDE	DROP_OFF_LATITUDE	PASSENGERS	DISTANCE	COST
3	2	2013-08-01 10:38:35	2013-08-01 10:38:51	N	0	0	0	0	1	.00	3.25
4	2	2013-08-01 11:51:45	2013-08-01 12:03:52	N	0	0	0	0	1	.00	8.5

Overwrite the `green_df` variable with the transforms performed on `tmp_df` in the previous step.

```
green_df = tmp_df
```

Perform the same transformation steps to the yellow taxi data.

```
tmp_df = (yellow_df
    .replace_na(columns=all_columns)
    .drop_nulls(*drop_if_all_null)
    .rename_columns(column_pairs={
        "vendor_name": "vendor",
        "VendorID": "vendor",
        "vendor_id": "vendor",
        "Trip_Pickup_DateTime": "pickup_datetime",
        "tpep_pickup_datetime": "pickup_datetime",
        "Trip_Dropoff_DateTime": "dropoff_datetime",
        "tpep_dropoff_datetime": "dropoff_datetime",
        "store_and_forward": "store_forward",
        "store_and_fwd_flag": "store_forward",
        "Start_Lon": "pickup_longitude",
        "Start_Lat": "pickup_latitude",
        "End_Lon": "dropoff_longitude",
        "End_Lat": "dropoff_latitude",
        "Passenger_Count": "passengers",
        "passenger_count": "passengers",
        "Fare_Amt": "cost",
        "fare_amount": "cost",
        "Trip_Distance": "distance",
        "trip_distance": "distance"
    })
    .keep_columns(columns=useful_columns))
tmp_df.head(5)
```

Again, overwrite `yellow_df` with `tmp_df`, and then call the `append_rows()` function on the green taxi data to append the yellow taxi data, creating a new combined dataframe.

```
yellow_df = tmp_df
combined_df = green_df.append_rows([yellow_df])
```

Convert types and filter

Examine the pickup and drop-off coordinates summary statistics to see how the data is distributed. First define a `TypeConverter` object to change the lat/long fields to decimal type. Next, call the `keep_columns()` function to restrict output to only the lat/long fields, and then call `get_profile()`.

```
decimal_type = dprep.TypeConverter(data_type=dprep.FieldType.DECIMAL)
combined_df = combined_df.set_column_types(type_conversions={
    "pickup_longitude": decimal_type,
    "pickup_latitude": decimal_type,
    "dropoff_longitude": decimal_type,
    "dropoff_latitude": decimal_type
})
combined_df.keep_columns(columns=[
    "pickup_longitude", "pickup_latitude",
    "dropoff_longitude", "dropoff_latitude"
]).get_profile()
```

From the summary statistics output, you see that there are coordinates that are missing, and coordinates that are not in New York City. Filter out coordinates not in the city border by chaining column filter commands within the `filter()` function, and defining minimum and maximum bounds for each field. Then call `get_profile()` again to verify the transformation.

```
tmp_df = (combined_df
    .drop_nulls(
        columns=["pickup_longitude", "pickup_latitude", "dropoff_longitude", "dropoff_latitude"],
        column_relationship=dprep.ColumnRelationship(dprep.ColumnRelationship.ANY)
    )
    .filter(dprep.f_and(
        dprep.col("pickup_longitude") <= -73.72,
        dprep.col("pickup_longitude") >= -74.09,
        dprep.col("pickup_latitude") <= 40.88,
        dprep.col("pickup_latitude") >= 40.53,
        dprep.col("dropoff_longitude") <= -73.72,
        dprep.col("dropoff_longitude") >= -74.09,
        dprep.col("dropoff_latitude") <= 40.88,
        dprep.col("dropoff_latitude") >= 40.53
    )))
tmp_df.keep_columns(columns=[
    "pickup_longitude", "pickup_latitude",
    "dropoff_longitude", "dropoff_latitude"
]).get_profile()
```

Parameter	Type	Min	Max	Count	Misaligned Count	Not Misaligned Count	Percentage Missing	Error Count	Empirical Count	0.1% Quantile	1% Quantile	5% Quantile	25% Quantile	50% Quantile	75% Quantile	95% Quantile	99% Quantile	99.9% Quantile	Standard Deviation	Mean	
PICKUP_LATITUDE	Flight	40.5	45.8	70	0.0	70	0.0	0.0	0.0	40.6	40.7	40.7	40.7	40.7	40.8	40.8	40.8	40.8	40.8	0.0	40.7
	Uptime	75.4	78.8	90	0.0	90	0.0	0.0	0.0	75.2	75.3	75.1	75.1	75.1	75.8	75.5	75.8	75.7	75.7	0.3	75.5
	Pilot	88.5	92.5	20	0.0	0.0	0.0	0.0	0.0	88.8	88.0	80.0	80.0	80.0	84.2	84.5	84.5	84.5	84.5	2.2	82.6
	Crew	55.0	58.0	20	0.0	0.0	0.0	0.0	0.0	54.4	54.5	50.0	50.0	50.0	53.2	53.5	53.5	53.5	53.5	0.0	52.6
	ICIM	44.0	48.0	50	0.0	0.0	0.0	0.0	0.0	44.0	40.0	0.0	0.0	0.0	40.0	40.0	40.0	40.0	40.0	0.0	40.0
	MA	55.0	58.0	20	0.0	0.0	0.0	0.0	0.0	54.4	54.5	50.0	50.0	50.0	53.2	53.5	53.5	53.5	53.5	0.0	52.6
	All	40.0	45.0	70	0.0	70	0.0	0.0	0.0	40.0	40.0	40.0	40.0	40.0	40.0	40.0	40.0	40.0	40.0	0.0	40.0
	Total	40.0	45.0	70	0.0	70	0.0	0.0	0.0	40.0	40.0	40.0	40.0	40.0	40.0	40.0	40.0	40.0	40.0	0.0	40.0
	Avg	40.0	45.0	70	0.0	70	0.0	0.0	0.0	40.0	40.0	40.0	40.0	40.0	40.0	40.0	40.0	40.0	40.0	0.0	40.0
	StdDev	40.0	45.0	70	0.0	70	0.0	0.0	0.0	40.0	40.0	40.0	40.0	40.0	40.0	40.0	40.0	40.0	40.0	0.0	40.0
DROP_LOCATION	Flight	-70.0	-74.0	50	0.0	70	0.0	0.0	0.0	-70.7	-70.7	-70.7	-70.7	-70.7	-70.7	-70.7	-70.7	-70.7	-70.7	0.0	-70.7
	Uptime	40.0	42.0	90	0.0	90	0.0	0.0	0.0	40.0	40.9	40.9	40.9	40.9	40.9	40.9	40.9	40.9	40.9	0.5	40.9
	Pilot	50.0	52.0	0	0.0	0.0	0.0	0.0	0.0	50.7	50.8	50.8	50.8	50.8	50.3	50.8	50.8	50.8	50.8	0.2	50.0
	Crew	47.0	48.0	1	0.0	0.0	0.0	0.0	0.0	48.0	46.6	48.8	48.8	48.8	46.0	48.0	48.5	48.5	48.5	0.1	47.1
	ICIM	47.0	48.0	1	0.0	0.0	0.0	0.0	0.0	48.0	46.6	48.8	48.8	48.8	46.0	48.0	48.5	48.5	48.5	0.1	47.1
	MA	47.0	48.0	1	0.0	0.0	0.0	0.0	0.0	48.0	46.6	48.8	48.8	48.8	46.0	48.0	48.5	48.5	48.5	0.1	47.1
	All	47.0	48.0	1	0.0	0.0	0.0	0.0	0.0	48.0	46.6	48.8	48.8	48.8	46.0	48.0	48.5	48.5	48.5	0.1	47.1
	Total	47.0	48.0	1	0.0	0.0	0.0	0.0	0.0	48.0	46.6	48.8	48.8	48.8	46.0	48.0	48.5	48.5	48.5	0.1	47.1
	Avg	47.0	48.0	1	0.0	0.0	0.0	0.0	0.0	48.0	46.6	48.8	48.8	48.8	46.0	48.0	48.5	48.5	48.5	0.1	47.1
	StdDev	47.0	48.0	1	0.0	0.0	0.0	0.0	0.0	48.0	46.6	48.8	48.8	48.8	46.0	48.0	48.5	48.5	48.5	0.1	47.1
DROP_FLOOR	Flight	40.0	45.0	70	0.0	70	0.0	0.0	0.0	40.0	40.0	40.0	40.0	40.0	40.0	40.0	40.0	40.0	40.0	0.0	40.0
	Uptime	30.0	35.0	50	0.0	50	0.0	0.0	0.0	30.0	34.0	33.0	33.0	33.0	33.0	33.0	33.0	33.0	33.0	0.5	33.0
	Pilot	50.0	52.0	0	0.0	0.0	0.0	0.0	0.0	50.0	50.0	50.0	50.0	50.0	50.0	50.0	50.0	50.0	50.0	0.0	50.0
	Crew	43.0	47.0	1	0.0	0.0	0.0	0.0	0.0	43.0	42.0	45.0	45.0	45.0	42.0	45.0	45.0	45.0	45.0	0.1	44.0
	ICIM	43.0	47.0	1	0.0	0.0	0.0	0.0	0.0	43.0	42.0	45.0	45.0	45.0	42.0	45.0	45.0	45.0	45.0	0.1	44.0
	MA	43.0	47.0	1	0.0	0.0	0.0	0.0	0.0	43.0	42.0	45.0	45.0	45.0	42.0	45.0	45.0	45.0	45.0	0.1	44.0
	All	43.0	47.0	1	0.0	0.0	0.0	0.0	0.0	43.0	42.0	45.0	45.0	45.0	42.0	45.0	45.0	45.0	45.0	0.1	44.0
	Total	43.0	47.0	1	0.0	0.0	0.0	0.0	0.0	43.0	42.0	45.0	45.0	45.0	42.0	45.0	45.0	45.0	45.0	0.1	44.0
	Avg	43.0	47.0	1	0.0	0.0	0.0	0.0	0.0	43.0	42.0	45.0	45.0	45.0	42.0	45.0	45.0	45.0	45.0	0.1	44.0
	StdDev	43.0	47.0	1	0.0	0.0	0.0	0.0	0.0	43.0	42.0	45.0	45.0	45.0	42.0	45.0	45.0	45.0	45.0	0.1	44.0
DROPLATITUDE	Flight	40.0	45.0	70	0.0	70	0.0	0.0	0.0	40.0	40.0	40.0	40.0	40.0	40.0	40.0	40.0	40.0	40.0	0.0	40.0
	Uptime	30.0	35.0	50	0.0	50	0.0	0.0	0.0	30.0	34.0	33.0	33.0	33.0	33.0	33.0	33.0	33.0	33.0	0.5	33.0
	Pilot	50.0	52.0	0	0.0	0.0	0.0	0.0	0.0	50.0	50.0	50.0	50.0	50.0	50.0	50.0	50.0	50.0	50.0	0.0	50.0
	Crew	43.0	47.0	1	0.0	0.0	0.0	0.0	0.0	43.0	42.0	45.0	45.0	45.0	42.0	45.0	45.0	45.0	45.0	0.1	44.0
	ICIM	43.0	47.0	1	0.0	0.0	0.0	0.0	0.0	43.0	42.0	45.0	45.0	45.0	42.0	45.0	45.0	45.0	45.0	0.1	44.0
	MA	43.0	47.0	1	0.0	0.0	0.0	0.0	0.0	43.0	42.0	45.0	45.0	45.0	42.0	45.0	45.0	45.0	45.0	0.1	44.0
	All	43.0	47.0	1	0.0	0.0	0.0	0.0	0.0	43.0	42.0	45.0	45.0	45.0	42.0	45.0	45.0	45.0	45.0	0.1	44.0
	Total	43.0	47.0	1	0.0	0.0	0.0	0.0	0.0	43.0	42.0	45.0	45.0	45.0	42.0	45.0	45.0	45.0	45.0	0.1	44.0
	Avg	43.0	47.0	1	0.0	0.0	0.0	0.0	0.0	43.0	42.0	45.0	45.0	45.0	42.0	45.0	45.0	45.0	45.0	0.1	44.0
	StdDev	43.0	47.0	1	0.0	0.0	0.0	0.0	0.0	43.0	42.0	45.0	45.0	45.0	42.0	45.0	45.0	45.0	45.0	0.1	44.0

Overwrite `combined_df` with the transformations you made to `tmp_df`.

```
combined_df = tmp_df
```

Split and rename columns

Look at the data profile for the `store_forward` column.

```
combined_df.keep_columns(columns='store_forward').get_profile()
```

From the data profile output of `store_forward`, you see that the data is inconsistent and there are missing/null values. Replace these values using the `replace()` and `fill_nulls()` functions, and in both cases change to the string "N".

```
combined_df = combined_df.replace(columns="store_forward", find="0",  
replace_with="N").fill_nulls("store_forward", "N")
```

Execute another `replace` function, this time on the `distance` field. This reformats distance values that are incorrectly labeled as `.00`, and fills any nulls with zeros. Convert the `distance` field to numerical format.

```
combined_df = combined_df.replace(columns="distance", find=".00", replace_with=0).fill_nulls("distance", 0)
combined_df = combined_df.to_number(["distance"])
```

Split the pick up and drop off datetimes into respective date and time columns. Use `split_column_by_example()` to perform the split. In this case, the optional `example` parameter of `split_column_by_example()` is omitted. Therefore the function will automatically determine where to split based on the data.

```

tmp_df = (combined_df
    .split_column_by_example(source_column="pickup_datetime")
    .split_column_by_example(source_column="dropoff_datetime"))
tmp_df.head(5)

```

		VEN DO R	PIC KU P_D ATE TIM E	PIC KU P_D ATE TIM E_1	PIC KU P_D ATE TIM E_2	DR OP OFF _DA TET IME	DR OP OFF _DA TET IME_1	DR OP OFF _DA TET IME_2	STO RE_ FOR WA RD	PIC KU P_L ON GIT UD E	PIC KU P_L ATI TU DE	DR OP OFF _LO NGI TU DE	DR OP OFF _LA TIT UD E	PAS SEN GER S	DIS TAN CE	COS T
0	2	20	20	17:	20	20	17:	N	-	40.	-	40.	1	0.0	2.5	
		13-	13-	22:	13-	13-	22:		73.	75	73.	75				
		08-	08-	00	08-	08-	00		93	84	93	84				
		01	01		01	01			77	80	77	80				
		17:			17:				67		67					
		22:			22:											
		00			00											
1	2	20	20	17:	20	20	17:	N	-	40.	-	40.	1	0.0	2.5	
		13-	13-	24:	13-	13-	25:		73.	75	73.	75				
		08-	08-	00	08-	08-	00		93	78	93	78				
		01	01		01	01			79	43	79	43				
		17:			17:				27		27					
		24:			25:											
		00			00											
2	2	20	20	06:	20	20	06:	N	-	40.	-	40.	1	0.0	3.3	
		13-	13-	51:	13-	13-	51:		73.	75	73.	75				
		08-	08-	19	08-	08-	36		93	84	93	83				
		06	06		06	06			77	04	77	69				
		06:			06:				21		21					
		51:			51:											
		19			36											
3	2	20	20	13:	20	20	13:	N	-	40.	-	40.	1	0.0	3.3	
		13-	13-	26:	13-	13-	26:		73.	75	73.	75				
		08-	08-	34	08-	08-	57		93	84	93	83				
		06	06		06	06			76	19	77	58				
		13:			13:				91		90					
		26:			26:											
		34			57											
4	2	20	20	13:	20	20	13:	N	-	40.	-	40.	1	0.0	3.3	
		13-	13-	27:	13-	13-	28:		73.	75	73.	75				
		08-	08-	53	08-	08-	08		93	83	93	84				
		06	06		06	06			78	96	77	50				
		13:			13:				05		75					
		27:			28:											
		53			08											

Rename the columns generated by `split_column_by_example()` into meaningful names.

```
tmp_df_renamed = (tmp_df
    .rename_columns(column_pairs={
        "pickup_datetime_1": "pickup_date",
        "pickup_datetime_2": "pickup_time",
        "dropoff_datetime_1": "dropoff_date",
        "dropoff_datetime_2": "dropoff_time"
    }))
tmp_df_renamed.head(5)
```

Overwrite `combined_df` with the executed transformations, and then call `get_profile()` to see full summary statistics after all transformations.

```
combined_df = tmp_df_renamed
combined_df.get_profile()
```

Transform data

Split the pickup and drop-off date further into day of week, day of month, and month. To get day of week, use the `derive_column_by_example()` function. This function takes as a parameter an array of example objects that define the input data, and the desired output. The function then automatically determines your desired transformation. For pickup and drop-off time columns, split into hour, minute, and second using the `split_column_by_example()` function with no example parameter.

Once you have generated these new features, delete the original fields in favor of the newly generated features using `drop_columns()`. Rename all remaining fields to accurate descriptions.

```

tmp_df = (combined_df
    .derive_column_by_example(
        source_columns="pickup_date",
        new_column_name="pickup_weekday",
        example_data=[("2009-01-04", "Sunday"), ("2013-08-22", "Thursday")]
    )
    .derive_column_by_example(
        source_columns="dropoff_date",
        new_column_name="dropoff_weekday",
        example_data=[("2013-08-22", "Thursday"), ("2013-11-03", "Sunday")]
    )
)

.split_column_by_example(source_column="pickup_time")
.split_column_by_example(source_column="dropoff_time")
# the following two split_column_by_example calls reference the generated column names from the above two
calls
.split_column_by_example(source_column="pickup_time_1")
.split_column_by_example(source_column="dropoff_time_1")
.drop_columns(columns=[
    "pickup_date", "pickup_time", "dropoff_date", "dropoff_time",
    "pickup_date_1", "dropoff_date_1", "pickup_time_1", "dropoff_time_1"
])
.rename_columns(column_pairs={
    "pickup_date_2": "pickup_month",
    "pickup_date_3": "pickup_monthday",
    "pickup_time_1_1": "pickup_hour",
    "pickup_time_1_2": "pickup_minute",
    "pickup_time_2": "pickup_second",
    "dropoff_date_2": "dropoff_month",
    "dropoff_date_3": "dropoff_monthday",
    "dropoff_time_1_1": "dropoff_hour",
    "dropoff_time_1_2": "dropoff_minute",
    "dropoff_time_2": "dropoff_second"
})))
tmp_df.head(5)

```


V	PIC	PIK	PIU	PIK	PIC	DRO	DRO	DRO	DRO	STORE	PICK	PICK	DROP	DROP	DROP	DROP	DROP	PASS	DI	C
E	KU	KU	PU	KU	CK	FF	FF	FF	FF	FOR	UP	KU	FF	FF	FF	FF	FF	NGE	ST	CO
N	TE	K	H	M	C	TE	A	H	M	GI	T	T	TI	TU	TI	TU	TI	NGE	ST	CO
D	TI	D	O	IN	O	TI	D	O	IN	W	U	A	U	D	U	D	DE	ER	NA	CE
O	M	A	U	U	N	M	A	U	U	R	D	R	D	E	D	E	DE	RS	NC	ST
R	E	Y	R	TE	D	E	Y	R	TE	D	D	D	D	E	D	E	DE	RS	NC	ST
3	2	2	T	1	2	3	2	T	1	2	5	N	-	4	-	4	1	0.	3.	
	0	0	u	3	6	4	0	u	3	6	7		7	0.	7	0.	0	0	3	
	1	1	es				1	es					3.	7	3.	7				
	3	3	d				3	d					9	5	9	5				
	-	-	a				-	a					3	8	3	8				
	0	0	y				0	y					7	4	7	3				
	8	8					8						6	1	7	5				
	-	-					-						9	9	9	8				
	0	0					0						1							
	6	6					6													
	1	1					1													
	3:	3:					3:													
	2	2					2													
	6:	6:					6:													
	3	3					5													
	4	4					7													
4	2	2	T	1	2	5	2	T	1	2	0	N	-	4	-	4	1	0.	3.	
	0	0	u	3	7	3	0	u	3	8	8		7	0.	7	0.	0	0	3	
	1	1	es				1	es					3.	7	3.	7				
	3	3	d				3	d					9	5	9	5				
	-	-	a				-	a					3	8	3	8				
	0	0	y				0	y					7	3	7	4				
	8	8					8						8	9	7	5				
	-	-					-						0	6	7	0				
	0	0					0						5							
	6	6					6													
	1	1					1													
	3:	3:					3:													
	2	2					2													
	7:	7:					8:													
	5	5					0													
	3	3					8													

From the data above, you see that the pickup and drop-off date and time components produced from the derived transformations are correct. Drop the `pickup_datetime` and `dropoff_datetime` columns as they are no longer needed.

```
tmp_df = tmp_df.drop_columns(columns=["pickup_datetime", "dropoff_datetime"])
```

Use the type inference functionality to automatically check the data type of each field, and display the inference results.

```
type_infer = tmp_df.builders.set_column_types()
type_infer.learn()
type_infer
```

```
Column types conversion candidates:  
'pickup_weekday': [FieldType.STRING],  
'pickup_hour': [FieldType.DECIMAL],  
'pickup_minute': [FieldType.DECIMAL],  
'pickup_second': [FieldType.DECIMAL],  
'dropoff_hour': [FieldType.DECIMAL],  
'dropoff_minute': [FieldType.DECIMAL],  
'dropoff_second': [FieldType.DECIMAL],  
'store_forward': [FieldType.STRING],  
'pickup_longitude': [FieldType.DECIMAL],  
'dropoff_longitude': [FieldType.DECIMAL],  
'passengers': [FieldType.DECIMAL],  
'distance': [FieldType.DECIMAL],  
'vendor': [FieldType.STRING],  
'dropoff_weekday': [FieldType.STRING],  
'pickup_latitude': [FieldType.DECIMAL],  
'dropoff_latitude': [FieldType.DECIMAL],  
'cost': [FieldType.DECIMAL]
```

The inference results look correct based on the data, now apply the type conversions to the dataflow.

```
tmp_df = type_infer.to_dataflow()  
tmp_df.get_profile()
```

Before packaging the dataflow, perform two final filters on the data set. To eliminate incorrect data points, filter the dataflow on records where both the `cost` and `distance` are greater than zero.

```
tmp_df = tmp_df.filter(dprep.col("distance") > 0)  
tmp_df = tmp_df.filter(dprep.col("cost") > 0)
```

At this point, you have a fully transformed and prepared dataflow object to use in a machine learning model. The SDK includes object serialization functionality, which is used as follows.

```
import os  
file_path = os.path.join(os.getcwd(), "dflows.dprep")  
  
dflow_prepared = tmp_df  
package = dprep.Package([dflow_prepared])  
package.save(file_path)
```

Clean up resources

Delete the file `dflows.dprep` (whether you are running locally or in Azure Notebooks) in your current directory if you do not wish to continue with part two of the tutorial. If you continue on to part two, you will need the `dflows.dprep` file in the current directory.

Next steps

In part one of this tutorial, you:

- Set up your development environment
- Loaded and cleansed data sets
- Used smart transforms to predict your logic based on an example
- Merged and packaged datasets for machine learning training

You are ready to use this training data in the next part of the tutorial series:

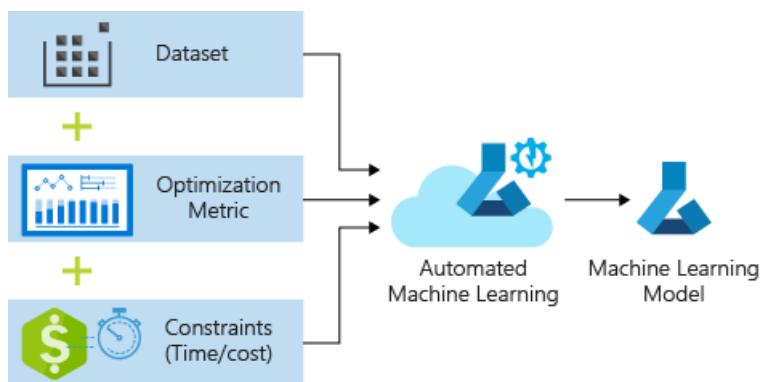
Tutorial #2: Train regression model

Tutorial: Use automated machine learning to build your regression model

12/12/2018 • 12 minutes to read • [Edit Online](#)

This tutorial is **part two of a two-part tutorial series**. In the previous tutorial, you [prepared the NYC taxi data for regression modeling](#).

Now, you're ready to start building your model with Azure Machine Learning service. In this part of the tutorial, you will use the prepared data and automatically generate a regression model to predict taxi fare prices. Using the automated ML capabilities of the service, you define your machine learning goals and constraints, launch the automated machine learning process and then allow the algorithm selection and hyperparameter-tuning to happen for you. The automated ML technique iterates over many combinations of algorithms and hyperparameters until it finds the best model based on your criterion.



In this tutorial, you learn how to:

- Setup a Python environment and import the SDK packages
- Configure an Azure Machine Learning service workspace
- Auto-train a regression model
- Run the model locally with custom parameters
- Explore the results
- Register the best model

If you don't have an Azure subscription, create a free account before you begin. Try the [free or paid version of Azure Machine Learning service](#) today.

NOTE

Code in this article was tested with Azure Machine Learning SDK version 1.0.0

Prerequisites

- [Run the data preparation tutorial](#).
- Automated machine learning configured environment e.g. Azure notebooks, Local Python environment or Data Science Virtual Machine. [Setup automated machine learning](#).

Get the notebook

For your convenience, this tutorial is available as a [Jupyter notebook](#). Run the `regression-part2-automated-ml.ipynb` notebook either in Azure Notebooks or in your own Jupyter notebook server.

Learn how to run notebooks by following the article, [Use Jupyter notebooks to explore this service](#).

Import packages

Import Python packages you need in this tutorial.

```
import azureml.core
import pandas as pd
from azureml.core.workspace import Workspace
from azureml.train.automl.run import AutoMLRun
import time
import logging
import os
```

Configure workspace

Create a workspace object from the existing workspace. A `Workspace` is a class that accepts your Azure subscription and resource information, and creates a cloud resource to monitor and track your model runs.

`Workspace.from_config()` reads the file **aml_config/config.json** and loads the details into an object named `ws`. `ws` is used throughout the rest of the code in this tutorial.

Once you have a workspace object, specify a name for the experiment and create and register a local directory with the workspace. The history of all runs is recorded under the specified experiment and in [Azure portal](#).

```
ws = Workspace.from_config()
# choose a name for the run history container in the workspace
experiment_name = 'automated-ml-regression'
# project folder
project_folder = './automated-ml-regression'

output = {}
output['SDK version'] = azureml.core.VERSION
output['Subscription ID'] = ws.subscription_id
output['Workspace'] = ws.name
output['Resource Group'] = ws.resource_group
output['Location'] = ws.location
output['Project Directory'] = project_folder
pd.set_option('display.max_colwidth', -1)
pd.DataFrame(data=output, index=['']).T
```

Explore data

Utilize the data flow object created in the previous tutorial. Open and execute the data flow and review the results.

```
import azureml.dataprep as dprep

file_path = os.path.join(os.getcwd(), "dflows.dprep")

package_saved = dprep.Package.open(file_path)
dflow_prepared = package_saved.dataflows[0]
dflow_prepared.get_profile()
```


		Statistical Summary																		Descriptive Statistics			
		Type	Min	Max	Count	Missing Count	Missing %	Percent Missing	Error Count	Empty Count	0% Quantile	1% Quantile	5% Quantile	25% Quantile	50% Quantile	75% Quantile	95% Quantile	99% Quantile	Mean	Standard Deviation	Variance	Skewness	Kurtosis
PICKUP-HOUR	Field Type	0	23	61	06148.	00000	00000	00000	00000	00000	00000	00000	00000	00000	00000	00000	00000	00000	142731	659242	-04623	-05723	-05704
	Field Type	0	59	61	06148.	00000	00000	00000	00000	00000	00000	00000	00000	00000	00000	00000	00000	00000	294273	30323	00120	01209	01208
	Field Type	0	59	61	06148.	00000	00000	00000	00000	00000	00000	00000	00000	00000	00000	00000	00000	00000	294273	30323	00120	01209	01208
	Field Type	0	59	61	06148.	00000	00000	00000	00000	00000	00000	00000	00000	00000	00000	00000	00000	00000	294273	30323	00120	01209	01208
	Field Type	0	59	61	06148.	00000	00000	00000	00000	00000	00000	00000	00000	00000	00000	00000	00000	00000	294273	30323	00120	01209	01208
	Field Type	0	59	61	06148.	00000	00000	00000	00000	00000	00000	00000	00000	00000	00000	00000	00000	00000	294273	30323	00120	01209	01208
	Field Type	0	59	61	06148.	00000	00000	00000	00000	00000	00000	00000	00000	00000	00000	00000	00000	00000	294273	30323	00120	01209	01208
	Field Type	0	59	61	06148.	00000	00000	00000	00000	00000	00000	00000	00000	00000	00000	00000	00000	00000	294273	30323	00120	01209	01208
	Field Type	0	59	61	06148.	00000	00000	00000	00000	00000	00000	00000	00000	00000	00000	00000	00000	00000	294273	30323	00120	01209	01208

Type	Min	Max	Count	Missing Count	Percent Missing	Error Count	Empty Count	0.1% Quantile	1% Quantile	5% Quantile	25% Quantile	50% Quantile	75% Quantile	95% Quantile	99% Quantile	99.9% Quantile	Mean	Standard Deviation	Variance	Skewness	Kurtosis
PICKUP-SECOND	Field Type . DECIMAL	0	59	614	0.48	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	2.9	1.7	3.0	-0.1	-1.1
DROPOFF-WEEKDAY	Field Type . STRING	Field Day	Wednesday	6148.	0.0	6148.	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	2.9	1.7	3.0	-0.1	-1.1

		Statistical Summary																		Descriptive Statistics			
		Type	Min	Max	Count	Missing Count	Missing %	Percent Missing	Error Count	Empty Count	0% Quantile	1% Quantile	5% Quantile	25% Quantile	50% Quantile	75% Quantile	95% Quantile	99% Quantile	Mean	Standard Deviation	Variance	Skewness	Kurtosis
DROPOFF-HOUR	Field Type	decimal	0	23	614	0.0	610	0.0	0.0	0.0	2.5	2.7	5.1	9.5	15.7	22.9	23.5	24.2	14.0	6.71	45.0	-0.6	-0.6
DROPOFF-MINUTE	Field Type	decimal	0	59	614	0.0	610	0.0	0.0	0.0	5.4	4.4	3.3	8.8	14.4	28.0	44.6	56.5	29.5	17.2	30.3	0.0	-1.1

Type	Min	Max	Count	Missing Count	Missing %	Percent Missing	Error Count	Empty Count	0.1 Quantile	1% Quantile	5% Quantile	25% Quantile	50% Quantile	75% Quantile	95% Quantile	99% Quantile	99.9% Quantile	Mean	Standard Deviation	Variance	Skewness	Kurtosis
PICKUP - LONGITUDE	Fidelity Type	-7.4	-7.4	6.1	0.0	6.1	0.0	0.0	-7.0	-7.4	-7.8	-7.3	-7.3	-7.3	-7.3	-7.3	-7.3	-7.7	-7.7	-0.0	0.3	-0.9
PICKUP - LATITUDE	Fidelity Type	4.0	4.0	6.1	0.0	6.1	0.0	0.0	4.0	4.0	4.0	4.0	4.0	4.0	4.0	4.0	4.0	4.0	4.0	0.0	0.0	-0.0

D R O P O F F _ L O N G I T U D E	T Y P E	M I N	M A X	C O U N T	M I S S I N G C O U N T	N O T M I S S I N G	P E R C E N T M I S S I N G	E R R O R C O U N T	E M P T Y C O U N T	0 . 1 % Q U A N T I L E	1 % Q U A N T I L E	5 % Q U A N T I L E	2 5 % Q U A N T I L E	5 0 % Q U A N T I L E	7 5 % Q U A N T I L E	9 5 % Q U A N T I L E	9 9 % Q U A N T I L E	9 9 . 9 % Q U A N T I L E	M E A N	S T A N D A R D D E V I A T I O N	V A R I A N C E	S K E W N E S S	K U R T O S I S		
																			M E A N	S T A N D A R D D E V I A T I O N	V A R I A N C E	S K E W N E S S	K U R T O S I S		
F i e l d T y p e	D E C I M A L	- 7 4 . 0 8	- 7 3 . 8	6 1 . 0	6 1 . 0	0 0 . 0	0 0 . 0	0 0 . 0	0 0 . 0	- 7 4 . 0	- 7 3 . 0	- 7 3 . 0	- 7 3 . 0	- 7 3 . 0	- 7 3 . 0	- 7 3 . 0	- 7 3 . 0	- 7 3 . 0	- 7 3 . 0	0 . 0	0 . 0	0 . 0	- 0 . 0		
F i e l d T y p e	D E C I M A L	4 5 . 7 0	4 9 . 9 9	6 1 . 0	6 1 . 0	0 0 . 0	0 0 . 0	0 0 . 0	0 0 . 0	- 7 7 . 5	- 7 7 . 5	- 7 5 . 2	- 8 8 . 8	- 9 3 . 3	- 9 6 . 3	- 9 9 . 5	- 9 5 . 5	- 9 5 . 5	- 9 5 . 5	- 9 5 . 9	- 7 2 . 7	- 7 2 . 7	- 7 2 . 7	- 7 8 . 9	- 0 . 3
D R O P O F F _ L A T I T U D E	D E C I M A L	4 0 . 5 8 . 3 5	4 0 . 7 9 . 7 5	6 1 . 0	6 1 . 0	0 0 . 0	0 0 . 0	0 0 . 0	0 0 . 0	- 4 0 . 5	- 4 0 . 5	- 4 0 . 5	- 4 0 . 5	- 4 0 . 5	- 4 0 . 5	- 4 0 . 5	- 4 0 . 5	- 4 0 . 5	- 4 0 . 5	- 4 0 . 5	0 . 0	0 . 0	0 . 0	- 0 . 0	
D R O P O F F _ L A T I T U D E	D E C I M A L	4 8 . 7 3 . 5	4 8 . 7 3 . 5	6 1 . 0	6 1 . 0	0 0 . 0	0 0 . 0	0 0 . 0	0 0 . 0	- 5 9 . 7	- 5 9 . 7	- 5 1 . 1	- 6 2 . 2	- 6 6 . 2	- 6 6 . 6	- 6 7 . 6	- 7 5 . 7	- 7 5 . 7	- 7 5 . 7	- 7 5 . 7	- 7 5 . 7	- 7 5 . 7	- 7 5 . 7	- 7 0 . 1	- 0 . 3

Type	Min	Max	Count	Total Missing Count	Percent Missing	Error Count	Empty Count	0.1% Quantile	1% Quantile	5% Quantile	25% Quantile	50% Quantile	75% Quantile	95% Quantile	99% Quantile	99.9% Quantile	Mean	Standard Deviation	Variance	Skewness	Kurtosis
PASSENGERS	Field Type	1	6	60	6.1408	0.0000	0.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	2.39249	1.83497	3.56144	0.76314	-1.23467
DISTANCE	Field Type	0	3	614080	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	3.10214	1.52943	1.85456	0.91555	4.98958

You prepare the data for the experiment by adding columns to `dflow_x` to be features for our model creation.

You define `dflow_y` to be our prediction value; cost.

```
dflow_X = dfflow_prepared.keep_columns(['pickup_weekday','pickup_hour', 'distance','passengers', 'vendor'])  
dflow_y = dfflow_prepared.keep_columns('cost')
```

Split data into train and test sets

Now you split the data into training and test sets using the `train_test_split` function in the `sklearn` library.

This function segregates the data into the x (features) data set for model training and the y (values to predict)

data set for testing. The `test_size` parameter determines the percentage of data to allocate to testing. The

`random_state` parameter sets a seed to the random generator, so that your train-test splits are always deterministic.

```
from sklearn.model_selection import train_test_split

x_df = df_X.to_pandas_dataframe()
y_df = df_y.to_pandas_dataframe()

x_train, x_test, y_train, y_test = train_test_split(x_df, y_df, test_size=0.2, random_state=223)
# flatten y_train to 1d array
y_train.values.flatten()
```

You now have the necessary packages and data ready for auto training for your model

Automatically train a model

To automatically train a model:

1. Define settings for the experiment run
2. Submit the experiment for model tuning

Define settings for autogeneration and tuning

Define the experiment parameters and models settings for autogeneration and tuning. View the full list of [settings](#).

PROPERTY	VALUE IN THIS TUTORIAL	DESCRIPTION
iteration_timeout_minutes	10	Time limit in minutes for each iteration
iterations	30	Number of iterations. In each iteration, the model trains with the data with a specific pipeline
primary_metric	spearman_correlation	Metric that you want to optimize.
preprocess	True	True enables experiment to perform preprocessing on the input.
verbosity	logging.INFO	Controls the level of logging.
n_cross_validationss	5	Number of cross validation splits

```
automl_settings = {  
    "iteration_timeout_minutes" : 10,  
    "iterations" : 30,  
    "primary_metric" : 'spearman_correlation',  
    "preprocess" : True,  
    "verbosity" : logging.INFO,  
    "n_cross_validations": 5  
}
```

```
from azureml.train.automl import AutoMLConfig  
  
# local compute  
automated_ml_config = AutoMLConfig(task = 'regression',  
                                    debug_log = 'automated_ml_errors.log',  
                                    path = project_folder,  
                                    X = x_train.values,  
                                    y = y_train.values.flatten(),  
                                    **automl_settings)
```

Train the automatic regression model

Start the experiment to run locally. Pass the defined `automated_ml_config` object to the experiment, and set the output to `True` to view progress during the experiment.

```
from azureml.core.experiment import Experiment  
experiment=Experiment(ws, experiment_name)  
local_run = experiment.submit(automated_ml_config, show_output=True)
```

```

Parent Run ID: AutoML_02778de3-3696-46e9-a71b-521c8fc0651
*****
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	MaxAbsScaler ExtremeRandomTrees	0:00:08	0.9447	0.9447
1	StandardScalerWrapper GradientBoosting	0:00:09	0.9536	0.9536
2	StandardScalerWrapper ExtremeRandomTrees	0:00:09	0.8580	0.9536
3	StandardScalerWrapper RandomForest	0:00:08	0.9147	0.9536
4	StandardScalerWrapper ExtremeRandomTrees	0:00:45	0.9398	0.9536
5	MaxAbsScaler LightGBM	0:00:08	0.9562	0.9562
6	StandardScalerWrapper ExtremeRandomTrees	0:00:27	0.8282	0.9562
7	StandardScalerWrapper LightGBM	0:00:07	0.9421	0.9562
8	MaxAbsScaler DecisionTree	0:00:08	0.9526	0.9562
9	MaxAbsScaler RandomForest	0:00:09	0.9355	0.9562
10	MaxAbsScaler SGD	0:00:09	0.9602	0.9602
11	MaxAbsScaler LightGBM	0:00:09	0.9553	0.9602
12	MaxAbsScaler DecisionTree	0:00:07	0.9484	0.9602
13	MaxAbsScaler LightGBM	0:00:08	0.9540	0.9602
14	MaxAbsScaler RandomForest	0:00:10	0.9365	0.9602
15	MaxAbsScaler SGD	0:00:09	0.9602	0.9602
16	StandardScalerWrapper ExtremeRandomTrees	0:00:49	0.9171	0.9602
17	SparseNormalizer LightGBM	0:00:08	0.9191	0.9602
18	MaxAbsScaler DecisionTree	0:00:08	0.9402	0.9602
19	StandardScalerWrapper ElasticNet	0:00:08	0.9603	0.9603
20	MaxAbsScaler DecisionTree	0:00:08	0.9513	0.9603
21	MaxAbsScaler SGD	0:00:08	0.9603	0.9603
22	MaxAbsScaler SGD	0:00:10	0.9602	0.9603
23	StandardScalerWrapper ElasticNet	0:00:09	0.9603	0.9603
24	StandardScalerWrapper ElasticNet	0:00:09	0.9603	0.9603
25	MaxAbsScaler SGD	0:00:09	0.9603	0.9603
26	TruncatedSVDWrapper ElasticNet	0:00:09	0.9602	0.9603
27	MaxAbsScaler SGD	0:00:12	0.9413	0.9603
28	StandardScalerWrapper ElasticNet	0:00:07	0.9603	0.9603
29	Ensemble	0:00:38	0.9622	0.9622

Explore the results

Explore the results of automatic training with a Jupyter widget or by examining the experiment history.

Option 1: Add a Jupyter widget to see results

If you are using a Jupyter notebook, use this Jupyter notebook widget to see a graph and a table of all results.

```

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

```



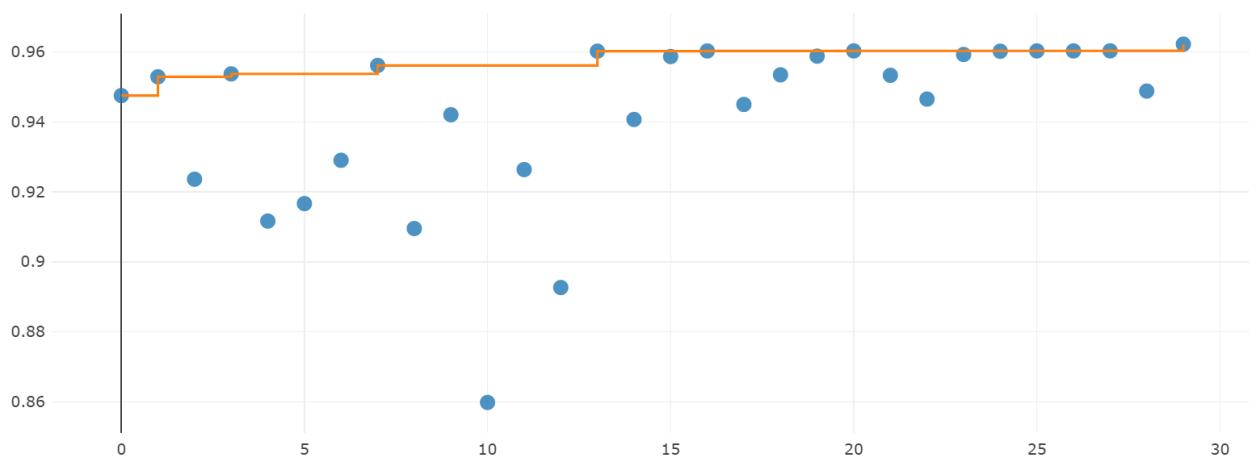
Iteration	Pipeline	Iteration metric	Best metric	Status	Duration	Started	Ran
29	Ensemble	0.96225654	0.96225654	Completed	0:01:10	Dec 6, 2018 6:12 PM	29
27	MaxAbsScaler, SGD	0.96033526	0.96033526	Completed	0:00:15	Dec 6, 2018 6:11 PM	27
25	StandardScalerWrapper, ElasticNet	0.96031892	0.96031892	Completed	0:00:11	Dec 6, 2018 6:11 PM	25
20	MaxAbsScaler, SGD	0.96031661	0.96031661	Completed	0:00:20	Dec 6, 2018 6:09 PM	20
26	StandardScalerWrapper, ElasticNet	0.96031391	0.96031892	Completed	0:00:12	Dec 6, 2018 6:11 PM	26

Pages: 1 2 3 4 5 6 Next Last 5 per page

spearman_correlation ▼



AutoML Run with metric : spearman_correlation



[Click here to see the run in Azure portal](#)

Option 2: Get and examine all run iterations in Python

Alternatively, you can retrieve the history of each experiment and explore the individual metrics for each iteration run.

```

children = list(local_run.get_children())
metricslist = {}
for run in children:
    properties = run.get_properties()
    metrics = {k: v for k, v in run.get_metrics().items() if isinstance(v, float)}
    metricslist[int(properties['iteration'])] = metrics

import pandas as pd
rundata = pd.DataFrame(metricslist).sort_index(1)
rundata

```


	0	1	2	3	4	5	6	7	8	9	...	20	21	22	23	24	25	26	27	28	29
NORMA LIZE D - ME DIAN_A BSOLU TE_ER OR	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0

	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0
	1	0	5	2	1	0	4	4	1	1	.	1	0	0	1	1	0	1	3	1	0
	6	9	2	0	8	9	8	1	0	5	.	1	8	9	3	2	8	2	0	1	9
	3	6	1	0	1	8	5	2	8	4	.	0	8	0	0	7	9	4	8	4	6
	6	8	0	8	4	9	3	6	6	8	.	7	1	7	5	0	1	9	1	1	9
	4	0	1	2	1	6	8	7	5	4	.	7	5	0	2	1	9	7	9	9	0
T_MEAN_SQ UARE_ER R	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0

	0	0	0	0	0	1	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0
	4	3	8	5	6	3	0	7	4	4	.	4	3	3	3	3	3	3	7	3	3
	7	7	5	2	5	8	9	1	2	9	.	2	7	7	7	7	7	7	2	7	6
	9	8	5	2	8	6	4	1	2	9	.	5	6	5	6	5	5	4	0	2	7
	6	8	7	8	0	6	0	0	9	6	.	6	8	5	4	1	6	6	7	4	1
	8	2	2	2	9	4	1	4	4	7	.	5	5	7	3	3	0	5	7	9	6

	0	1	2	3	4	5	6	7	8	9	...	20	21	22	23	24	25	26	27	28	29
NORMA LIZE D_R OO T_M EAN_S QUA RE D_L OG_E RROR	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0

	0	0	1	0	0	0	1	0	0	0	...	0	0	0	0	0	0	0	0	0	0
	5	4	1	6	6	4	2	9	4	5	...	4	4	4	4	4	4	4	7	4	4
	5	5	0	5	3	4	3	2	6	5	...	6	1	1	5	4	1	4	9	2	1
	3	0	2	6	5	4	4	3	1	2	...	5	8	7	1	6	6	4	6	7	5
	5	0	1	3	8	1	3	1	3	4	...	4	0	7	7	2	1	0	5	9	3
	3	0	9	3	9	2	3	2	0	3	...	0	4	1	5	8	7	5	1	9	0
R2_SCO RE	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0

	8	8	3	7	6	8	0	5	8	7	...	8	8	8	8	8	8	8	5	8	8
	1	8	9	7	4	7	2	8	5	9	...	4	8	8	8	8	8	8	4	8	8
	0	0	8	5	2	5	1	6	1	3	...	9	0	0	0	1	0	1	8	2	6
	9	3	0	9	8	7	6	5	7	6	...	8	1	9	5	3	8	6	1	8	3
	0	2	7	5	1	1	0	1	6	7	...	0	4	5	8	4	8	1	2	8	2
	0	8	6	7	2	9	3	4	7	1	...	9	2	2	6	7	7	3	1	3	1
ROOT_M EAN_S QUA RE D_E RROR	4	3	7	4	5	3	9	6	3	4	...	3	3	3	3	3	3	3	6	3	3

	2	3	5	5	7	3	6	2	7	3	...	7	3	3	3	2	3	2	3	2	2
	1	2	2	9	8	9	1	5	1	9	...	4	1	0	0	9	0	9	3	7	2
	6	9	1	5	4	8	6	0	7	2	...	1	2	1	8	7	1	3	5	4	7
	3	8	7	6	6	5	3	0	6	0	...	4	5	2	7	3	4	1	5	2	3
	6	1	6	0	0	4	5	1	6	7	...	4	3	4	9	8	8	8	0	6	5
	2	0	5	4	1	0	4	1	1	2	...	7	3	2	5	9	5	2	1	9	5

	0	1	2	3	4	5	6	7	8	9	...	20	21	22	23	24	25	26	27	28	29
R O O T_ M E A N _S Q U A R E D _L O G _E R R O R	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0

	2	1	4	2	2	1	5	4	2	2	.	2	1	1	1	1	1	1	3	1	1
	4	9	8	8	7	9	4	0	0	4	.	0	8	8	9	9	8	9	4	8	8
	3	7	4	8	9	5	2	5	2	2	.	4	3	3	8	6	2	5	9	8	2
	1	7	2	3	3	1	2	5	6	7	.	4	6	5	4	0	8	0	9	0	4
	8	0	2	4	6	1	8	5	6	0	.	6	5	1	6	6	3	8	3	3	5
	4	2	7	9	7	6	1	9	6	2	.	4	8	4	8	7	6	7	5	1	5
S P E A R M A N _C O R R E L A T I O N	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0

	9	9	8	9	9	9	8	9	9	9	.	9	9	9	9	9	9	9	9	9	9
	4	5	5	1	3	5	2	4	5	3	.	5	6	6	6	6	6	6	4	6	6
	4	3	7	4	9	6	8	2	2	5	.	1	0	0	0	0	0	0	1	0	2
	7	6	9	7	8	1	1	0	5	4	.	2	3	1	2	2	3	1	2	2	1
	4	1	6	0	4	5	8	6	8	7	.	8	3	9	7	8	2	6	5	9	5
	3	8	5	3	6	9	7	9	1	7	.	7	5	5	9	8	3	1	4	3	8
S P E A R M A N _C O R R E L A T I O N - M A X	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0

	9	9	9	9	9	9	9	9	9	9	.	9	9	9	9	9	9	9	9	9	9
	4	5	5	5	5	5	5	5	5	5	.	6	6	6	6	6	6	6	6	6	6
	4	3	3	3	3	6	6	6	6	6	.	0	0	0	0	0	0	0	0	0	2
	7	6	6	6	6	1	1	1	1	1	.	3	3	3	3	3	3	3	3	3	1
	4	1	1	1	1	5	5	5	5	5	.	0	3	3	3	3	3	3	3	3	5
	3	8	8	8	8	9	9	9	9	9	.	3	5	5	5	5	5	5	5	5	8

12 rows × 30 columns

Retrieve the best model

Select the best pipeline from our iterations. The `get_output` method on `automl_classifier` returns the best run and the fitted model for the last fit invocation. There are overloads on `get_output` that allow you to retrieve the best run and fitted model for any logged metric or a particular iteration.

```
best_run, fitted_model = local_run.get_output()
print(best_run)
print(fitted_model)
```

Register the model

Register the model in your Azure Machine Learning service workspace.

```
description = 'Automated Machine Learning Model'
tags = None
local_run.register_model(description=description, tags=tags)
local_run.model_id # Use this id to deploy the model as a web service in Azure
```

Test the best model accuracy

Use the best model to run predictions on the test data set. The function `predict` uses the best model, and predicts the values of `y` (trip cost) from the `x_test` data set. Print the first 10 predicted cost values from `y_predict`.

```
y_predict = fitted_model.predict(x_test.values)
print(y_predict[:10])
```

Create a scatter plot to visualize the predicted cost values compared to the actual cost values. The following code uses the `distance` feature as the x-axis, and trip `cost` as the y-axis. The first 100 predicted and actual cost values are created as separate series, in order to compare the variance of predicted cost at each trip distance. Examining the plot shows that the distance/cost relationship is nearly linear, and the predicted cost values are in most cases very close to the actual cost values for the same trip distance.

```
import matplotlib.pyplot as plt

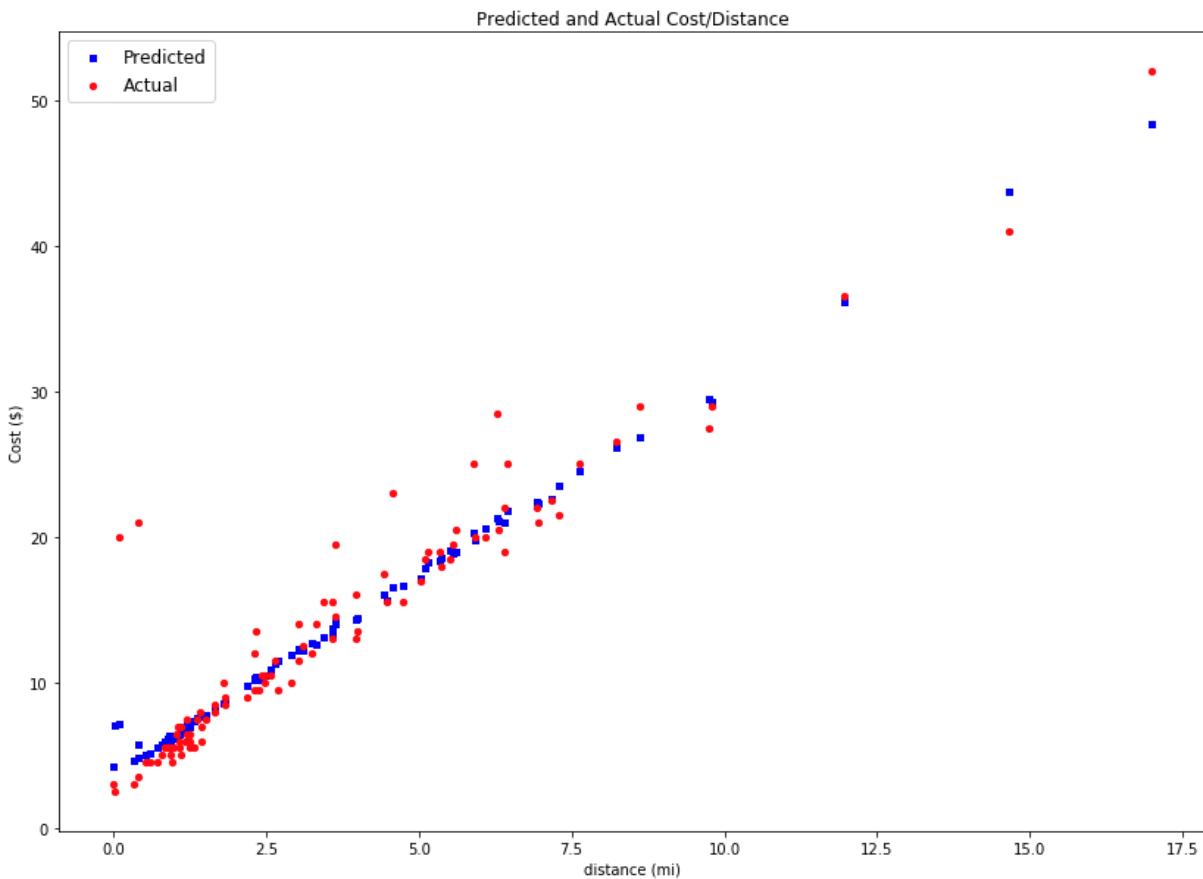
fig = plt.figure(figsize=(14, 10))
ax1 = fig.add_subplot(111)

distance_vals = [x[4] for x in x_test.values]
y_actual = y_test.values.flatten().tolist()

ax1.scatter(distance_vals[:100], y_predict[:100], s=18, c='b', marker="s", label='Predicted')
ax1.scatter(distance_vals[:100], y_actual[:100], s=18, c='r', marker="o", label='Actual')

ax1.set_xlabel('distance (mi)')
ax1.set_title('Predicted and Actual Cost/Distance')
ax1.set_ylabel('Cost ($')

plt.legend(loc='upper left', prop={'size': 12})
plt.rcParams.update({'font.size': 14})
plt.show()
```



Calculate the `root mean squared error` of the results. Use the `y_test` dataframe, and convert it to a list to compare to the predicted values. The function `mean_squared_error` takes two arrays of values, and calculates the average squared error between them. Taking the square root of the result gives an error in the same units as the `y` variable (cost), and indicates roughly how far your predictions are from the actual value.

```
from sklearn.metrics import mean_squared_error
from math import sqrt

rmse = sqrt(mean_squared_error(y_actual, y_predict))
rmse
```

```
3.2204936862688798
```

Run the following code to calculate MAPE (mean absolute percent error) using the full `y_actual` and `y_predict` data sets. This metric calculates an absolute difference between each predicted and actual value, sums all the differences, and then expresses that sum as a percent of the total of the actual values.

```

sum_actuals = sum_errors = 0

for actual_val, predict_val in zip(y_actual, y_predict):
    abs_error = actual_val - predict_val
    if abs_error < 0:
        abs_error = abs_error * -1

    sum_errors = sum_errors + abs_error
    sum_actuals = sum_actuals + actual_val

mean_abs_percent_error = sum_errors / sum_actuals
print("Model MAPE:")
print(mean_abs_percent_error)
print()
print("Model Accuracy:")
print(1 - mean_abs_percent_error)

```

Model MAPE:
0.10545153869569586

Model Accuracy:
0.8945484613043041

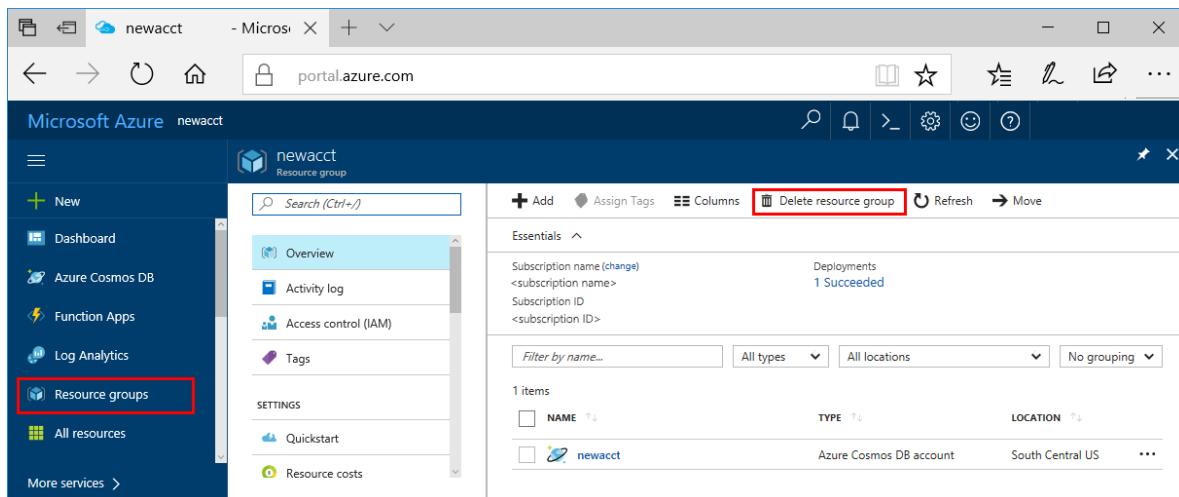
Clean up resources

IMPORTANT

The resources you created can be used as prerequisites to other Azure Machine Learning service tutorials and how-to articles.

If you don't plan to use the resources you created here, delete them so you don't incur any charges.

1. In the Azure portal, select **Resource groups** on the far left.



2. From the list, select the resource group you created.
3. Select **Delete resource group**.
4. Enter the resource group name, and then select **Delete**.

Next steps

In this automated machine learning tutorial, you:

- Configured a workspace and prepared data for an experiment
- Trained using an automated regression model locally with custom parameters
- Explored and reviewed training results
- Registered the best model

[Deploy your model](#) with Azure Machine Learning.

Use Jupyter notebooks to explore Azure Machine Learning service

12/7/2018 • 3 minutes to read • [Edit Online](#)

For your convenience, we have developed a series of Jupyter Python notebooks you can use to explore the Azure Machine Learning service.

Learn how to use the service with the documentation on this site and use these notebooks to customize them to your situation.

Prerequisite

Complete the [Azure Machine Learning Python quickstart](#) to create a workspace and launch Azure Notebooks.

Try Azure Notebooks: Free Jupyter notebooks in the cloud

It's easy to get started with Azure Notebooks! The [Azure Machine Learning SDK for Python](#) is already installed and configured for you on Azure Notebooks. The installation and future updates are automatically managed via Azure services.

- To run the **core tutorial notebooks**:

1. Go to [Azure Notebooks](#).
2. Find the **tutorials** folder in the **Getting Started** library you created during the prerequisite quickstart.
3. Open the notebook you want to run.

- To run **other notebooks**:

1. [Import the sample notebooks](#) into Azure Notebooks.
2. Add a workspace configuration file to the library using either of these methods:
 - Copy the **config.json** file from the **Getting Started** Library into the new cloned library.
 - Create a new workspace using code in the [configuration.ipynb](#).
3. Open the notebook you want to run.

Use a Data Science Virtual Machine (DSVM)

The [Azure Machine Learning SDK for Python](#) and notebook server are already installed and configured for you on a DSVM. Use these steps run the notebooks.

1. [Create a DSVM](#).
2. Clone [the GitHub repository](#).
3. Add a workspace configuration file to the library using either of these methods:
 - Copy the **aml_config\config.json** file you created using the prerequisite quickstart into the cloned directory.

- Create a new workspace using code in the [configuration.ipynb](#).
4. Start the notebook server from your cloned directory.

Use your own Jupyter notebook server

Use these steps to create a local Jupyter Notebook server on your computer.

1. Ensure you've completed the prerequisite quickstart in which you installed the Azure Machine Learning SDKs.
2. Clone [the GitHub repository](#).
3. Add a workspace configuration file to the library using either of these methods:
 - Copy the **aml_config\config.json** file you created using the prerequisite quickstart into the cloned directory.
 - Create a new workspace using code in the [configuration.ipynb](#).
4. Start the notebook server from your cloned directory.
5. Go to the folder containing the notebook.
6. Open the notebook.

Automated ML setup

These steps apply only to the notebooks in the `automated-machine-learning` folder.

While you can use any of the above options, you can also install the environment and create a workspace at the same time with the following instructions.

1. Install [Mini-conda](#). Choose 3.7 or higher. Follow prompts to install.

NOTE

You can use an existing conda as long as it is version 4.4.10 or later. Use `conda -v` to display the version. You can update a conda version with the command: `conda update conda`. There's no need to install mini-conda specifically.
2. Download the sample notebooks from [Github](#) as a zip and extract the contents to a local directory. The Automated machine learning notebooks are in the `how-to-use-azureml/automated-machine-learning` folder.
3. Set up a new Conda environment.
 - a. Open a Conda prompt on your local machine.
 - b. Navigate to the files you extracted to your local machine.
 - c. Open the `automated-machine-learning` folder.
 - d. Execute `automl_setup.cmd` in the conda prompt for Windows, or the `.sh` file for your operating system. It can take about 10 minutes to execute.

The setup script:

- Creates a new conda environment
- Installs the necessary packages
- Configures the widget

- Starts a jupyter notebook

The script takes the conda environment name as an optional parameter. The default conda environment name is `azure_automl`. The exact command depends on the operating system.

Once the script has completed, you will see a Jupyter notebook home page in your browser.

4. Navigate to the path where you saved the notebooks.
5. Open the automated-machine-learning folder, then open the `configuration.ipynb` notebook.
6. Execute the cells in the notebook to register Machine Learning Services Resource Provider and create a workspace.

You are now ready to open and run the notebooks saved on your local machine.

Next steps

Explore the [GitHub notebooks repository for Azure Machine Learning service](#)

Try these tutorials:

- [Train and deploy an image classification model with MNIST](#)
- [Prepare data and use automated machine learning to train a regression model with the NYC taxi data set](#)

What is automated machine learning?

12/11/2018 • 2 minutes to read • [Edit Online](#)

Automated machine learning is the process of taking training data with a defined target feature, and iterating through combinations of algorithms and feature selections to automatically select the best model for your data based on the training scores. The traditional machine learning model development process is highly resource-intensive, and requires significant domain knowledge and time investment to run and compare the results of dozens of models. Automated machine learning simplifies this process by generating models tuned from the goals and constraints you defined for your experiment, such as the time for the experiment to run or which models to blacklist.

How it works

1. You configure the type of machine learning problem you are trying to solve. Categories of supervised learning are supported:

- Classification
- Regression
- Forecasting

While automated machine learning is generally available, **the forecasting feature is still in public preview.**

See the [list of models](#) Azure Machine Learning can try when training.

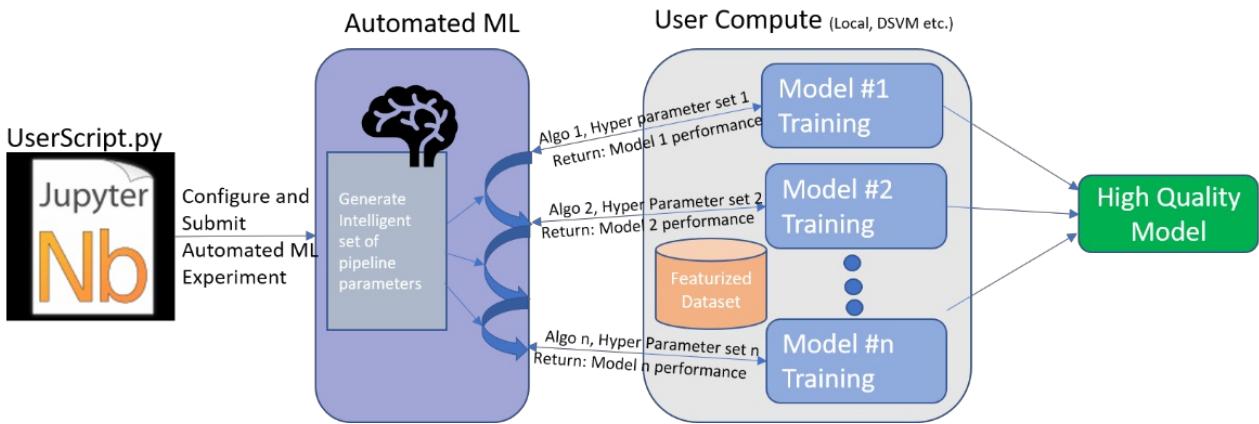
2. You specify the source and format for the training data. The data must be labeled, and can be stored on your development environment or in Azure Blob Storage. If the data is stored on your development environment, it must be in the same directory as your training scripts. This directory is copied to the compute target you select for training.

In your training script, the data can be read into Numpy arrays or a Pandas dataframe.

You can configure split options for selecting training and validation data, or you can specify separate training and validation data sets.

3. Configure the [compute target](#) that is used to train the model.
4. Configure the automated machine learning configuration. This controls the parameters used as Azure Machine Learning iterates over different models, hyperparameter settings, and what metrics to look at when determining the best model
5. Submit a training run.

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.



You can inspect the logged run information, which contains metrics gathered during the run. The training run also produces a Python serialized object (`.pk1` file) that contains the model and data preprocessing.

Model explainability

A common pitfall of automated machine learning is an inability to see the end-to-end process. Azure Machine Learning allows you to view detailed information about the models to increase transparency into what's running on the back-end. Output shows overall feature importance in model tuning, ranking the results by the features that influenced your model the most. Additionally, for classification problems you can see the per-class feature importance and ranking.

Next steps

See examples and learn how to build models using Automated Machine Learning:

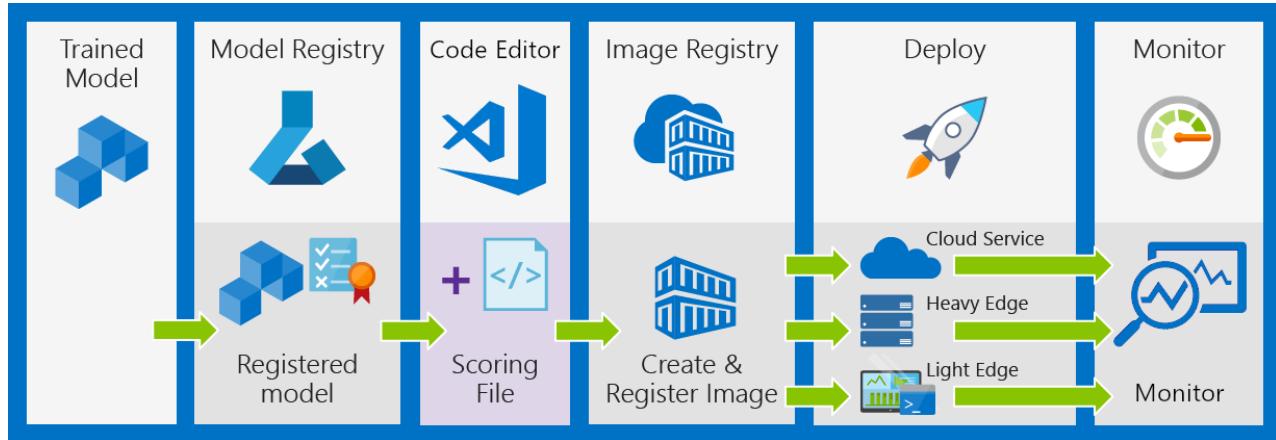
- [Tutorial: Automatically train a classification model with Azure Automated Machine Learning](#)
- [Configure settings for automatic training](#)
- [Use automatic training on a remote resource](#)

Manage, deploy, and monitor models with Azure Machine Learning Service

12/11/2018 • 3 minutes to read • [Edit Online](#)

In this article, you can learn how to use Azure Machine Learning Service to deploy, manage, and monitor your models to continuously improve them. You can deploy the models you trained with Azure Machine Learning, on your local machine, or from other sources.

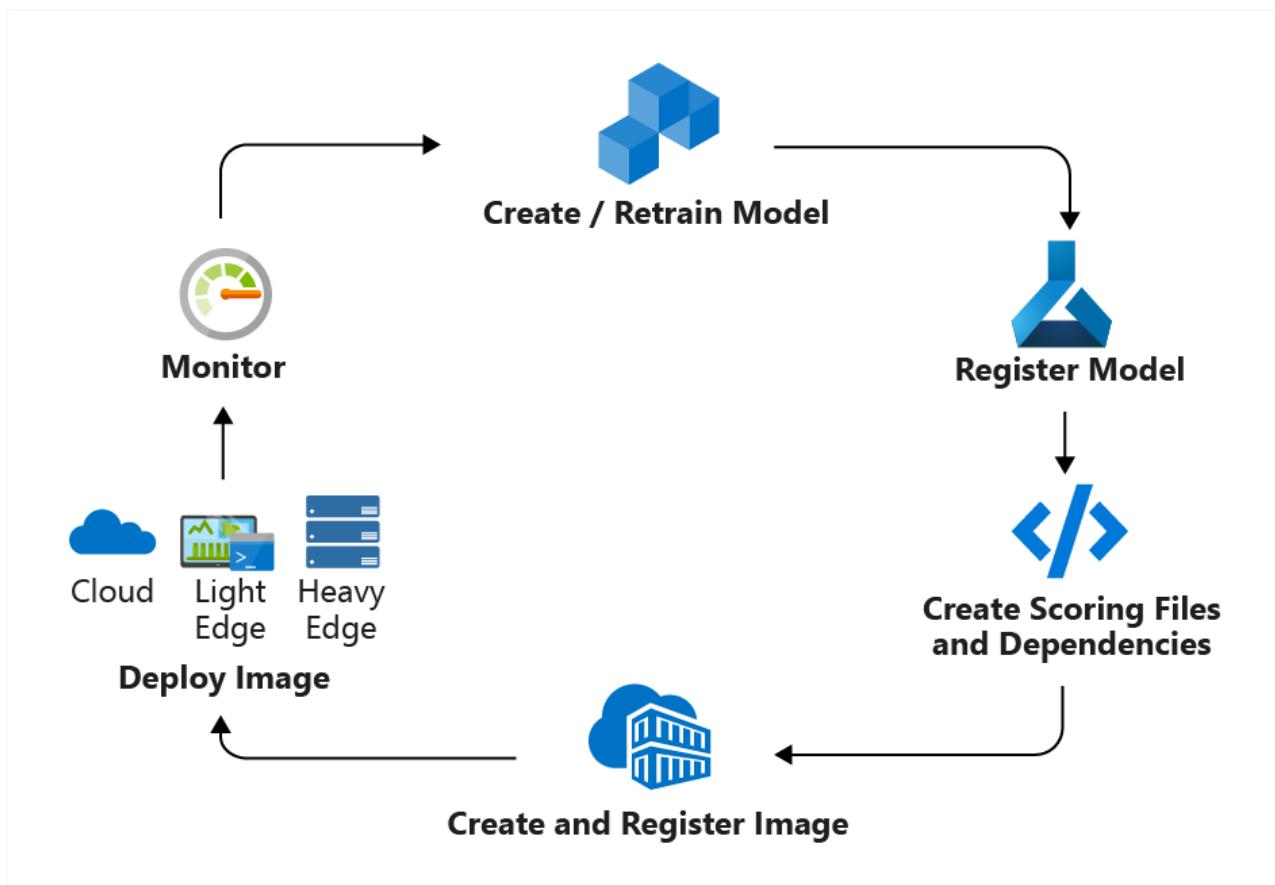
The following diagram illustrates the complete deployment workflow:



The deployment workflow includes the following steps:

1. **Register the model** in a registry hosted in your Azure Machine Learning Service workspace
2. **Register an image** that pairs a model with a scoring script and dependencies in a portable container
3. **Deploy** the image as a web service in the cloud or to edge devices
4. **Monitor and collect data**

Each step can be performed independently or as part of a single deployment command. Additionally, you can integrate deployment into a **CI/CD workflow** as illustrated in this graphic.



Step 1: Register model

The model registry keeps track of all the models in your Azure Machine Learning Service workspace. Models are identified by name and version. Each time you register a model with the same name as an existing one, the registry increments the version. You can also provide additional metadata tags during registration that can be used when searching for models.

You can't delete models that are being used by an image.

Step 2: Register image

Images allow for reliable model deployment, along with all components needed to use the model. An image contains the following items:

- The model
- The scoring engine
- The scoring file or application
- Any dependencies needed to score the model

The image can also include SDK components for logging and monitoring. The SDK logs data can be used to fine-tune or retrain your model, including the input and output of the model.

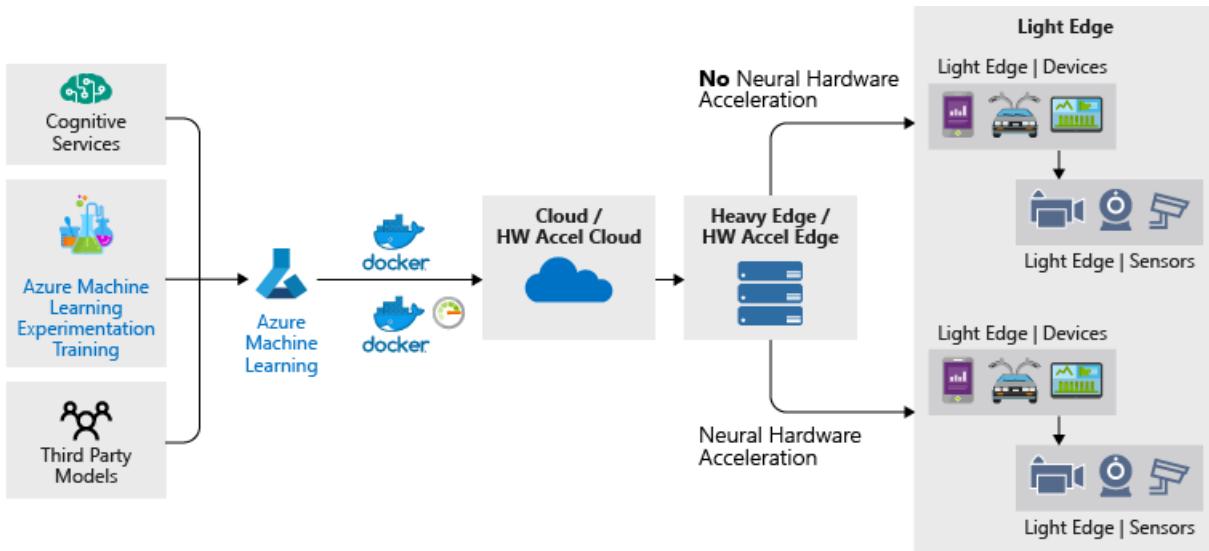
Azure Machine Learning supports the most popular frameworks, but in general any framework that can be pip installed can work.

When your workspace was created, so were other several other Azure resources used by that workspace. All the objects used to create the image are stored in the Azure storage account in your workspace. The image is created and stored in the Azure Container Registry. You can provide additional metadata tags when creating the image, which are also stored by the image registry and can be queried to find your image.

Step 3: Deploy image

You can deploy registered images into the cloud or to edge devices. The deployment process creates all the resources needed to monitor, load-balance, and auto-scale your model. Access to the deployed services can be secured with certificate based authentication by providing the security assets during deployment. You can also upgrade an existing deployment to use a newer image.

Web service deployments are also searchable. For example, you can search for all deployments of a specific model or image.



You can deploy your images to the following [deployment targets](#) in the cloud:

- Azure Container Instance
- Azure Kubernetes Service
- Azure FPGA machines
- Azure IoT Edge devices

As your service is deployed, the inferencing request is automatically load-balanced and the cluster is scaled to satisfy any spikes on demand. [Telemetry about your service can be captured](#) into the Azure Application Insights service associated with your Workspace.

Step 4: Monitor models and collect data

An SDK for model logging and data capture is available so you can monitor input, output, and other relevant data from your model. The data is stored as a blob in the Azure Storage account for your workspace.

To use the SDK with your model, you import the SDK into your scoring script or application. You can then use the SDK to log data such as parameters, results, or input details.

If you decide to [enable model data collection](#) every time you deploy the image, the details needed to capture the data, such as the credentials to your personal blob store, are provisioned automatically.

IMPORTANT

Microsoft does not see the data you collect from your model. The data is sent directly to the Azure storage account for your workspace.

Next steps

Learn more about [how and where you can deploy models](#) with the Azure Machine Learning service.

What are FPGAs and Project Brainwave?

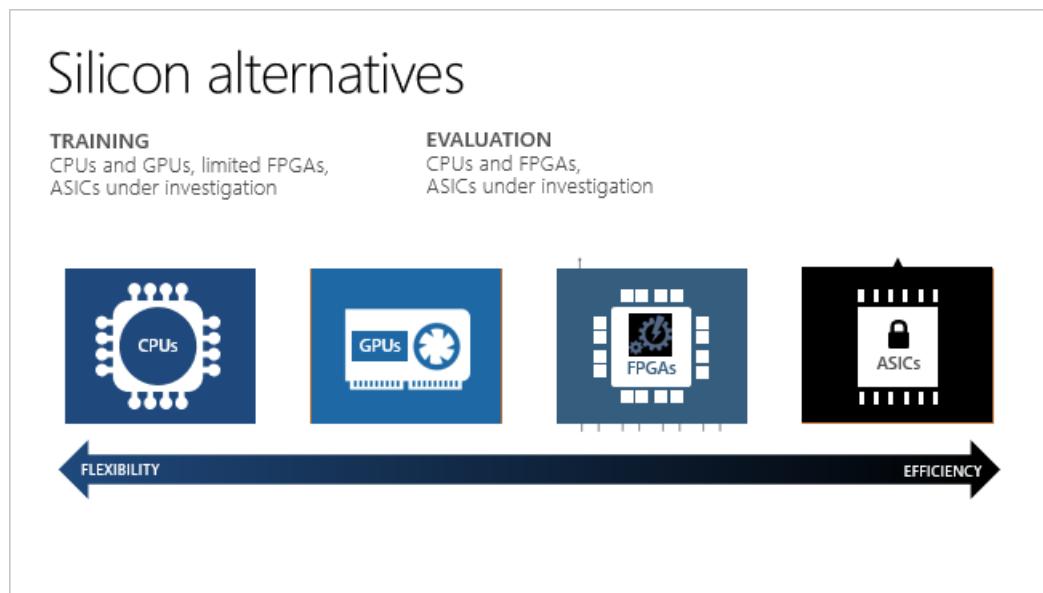
12/12/2018 • 3 minutes to read • [Edit Online](#)

This article provides an introduction to field-programmable gate arrays (FPGA), and how the Azure Machine Learning service provides real-time artificial intelligence (AI) when you deploy your model to an Azure FPGA.

FPGAs contain an array of programmable logic blocks, and a hierarchy of reconfigurable interconnects. The interconnects allow these blocks to be configured in various ways after manufacturing. Compared to other chips, FPGAs provide a combination of programmability and performance.

FPGAs vs. CPU, GPU, and ASIC

The following diagram and table show how FPGAs compare to other processors.



PROCESSOR		DESCRIPTION
Application-specific integrated circuits	ASICs	Custom circuits, such as Google's TensorFlow Processor Units (TPU), provide the highest efficiency. They can't be reconfigured as your needs change.
Field-programmable gate arrays	FPGAs	FPGAs, such as those available on Azure, provide performance close to ASICs. They are also flexible and reconfigurable over time, to implement new logic.
Graphics processing units	GPUs	A popular choice for AI computations. GPUs offer parallel processing capabilities, making it faster at image rendering than CPUs.
Central processing units	CPUs	General-purpose processors, the performance of which isn't ideal for graphics and video processing.

Project Brainwave on Azure

[Project Brainwave](#) is a hardware architecture from Microsoft. It's based on Intel's FPGA devices, which data scientists and developers use to accelerate real-time AI calculations. This FPGA-enabled architecture offers performance, flexibility, and scale, and is available on Azure.

FPGAs make it possible to achieve low latency for real-time inferencing requests. Asynchronous requests (batching) aren't needed. Batching can cause latency, because more data needs to be processed. Project Brainwave implementations of neural processing units don't require batching; therefore the latency can be many times lower, compared to CPU and GPU processors.

Reconfigurable power

You can reconfigure FPGAs for different types of machine learning models. This flexibility makes it easier to accelerate the applications based on the most optimal numerical precision and memory model being used. Because FPGAs are reconfigurable, you can stay current with the requirements of rapidly changing AI algorithms.

What's supported on Azure

Microsoft Azure is the world's largest cloud investment in FPGAs. You can run Project Brainwave on Azure's scale infrastructure.

Today, Project Brainwave supports:

- Image classification and recognition scenarios
- TensorFlow deployment
- DNNs: ResNet 50, ResNet 152, VGG-16, SSD-VGG, and DenseNet-121
- Intel FPGA hardware

Using this FPGA-enabled hardware architecture, trained neural networks run quickly and with lower latency. Project Brainwave can parallelize pre-trained deep neural networks (DNN) across FPGAs to scale out your service. The DNNs can be pre-trained, as a deep featurizer for transfer learning, or fine-tuned with updated weights.

Scenarios and applications

Project Brainwave is integrated with Azure Machine Learning. Microsoft uses FPGAs for DNN evaluation, Bing search ranking, and software defined networking (SDN) acceleration to reduce latency, while freeing CPUs for other tasks.

The following scenarios use FPGA on Project Brainwave architecture:

- [Automated optical inspection system](#)
- [Land cover mapping](#)

Deploy to FPGAs on Azure

To create an image recognition service in Azure, you can use supported DNNs as a featurizer for deployment on Azure FPGAs:

1. Use the [Azure Machine Learning SDK for Python](#) to create a service definition. A service definition is a file describing a pipeline of graphs (input, featurizer, and classifier) based on TensorFlow. The deployment command automatically compresses the definition and graphs into a ZIP file, and uploads the ZIP to Azure Blob storage. The DNN is already deployed on Project Brainwave to run on the FPGA.
2. Register the model by using the SDK with the ZIP file in Azure Blob storage.
3. Deploy the service with the registered model by using the SDK.

To get started deploying trained DNN models to FPGAs in the Azure cloud, see [Deploy a model as a web service](#)

on an FPGA.

Next steps

Check out these videos and blogs:

- [Hyperscale hardware: ML at scale on top of Azure + FPGA : Build 2018 \(video\)](#)
- [Inside the Microsoft FPGA-based configurable cloud \(video\)](#)
- [Project Brainwave for real-time AI: project home page](#)

Build machine learning pipelines with the Azure Machine Learning service

12/12/2018 • 3 minutes to read • [Edit Online](#)

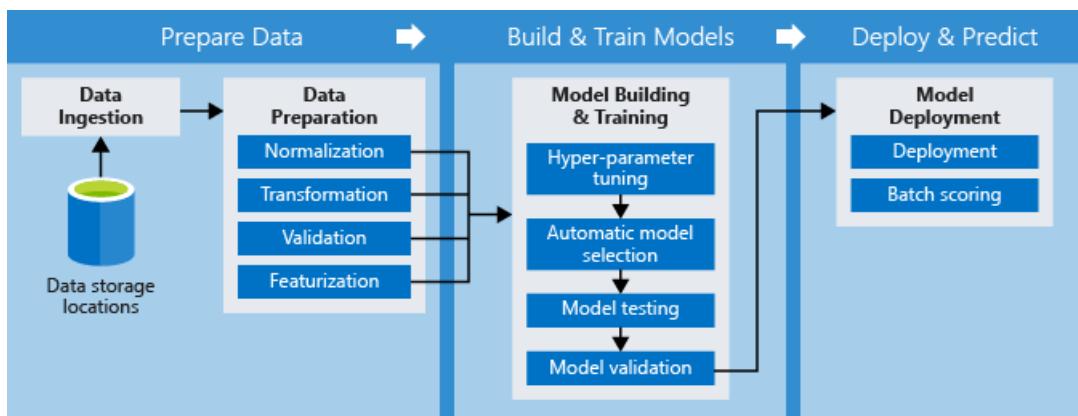
In this article, learn about the machine learning pipelines you can build with the Azure Machine Learning SDK for Python, and the advantages to using pipelines.

What are machine learning pipelines?

Using machine learning (ML) pipelines, data scientists, data engineers, and IT professionals can collaborate on the steps involved in:

- Data preparation, such as normalizations and transformations
- Model training
- Model evaluation
- Deployment

The following diagram shows an example pipeline:



Why build pipelines with Azure Machine Learning?

You can use the [Azure Machine Learning SDK for Python](#) to create ML pipelines, as well as to submit and track individual pipeline runs.

With pipelines, you can optimize your workflow with simplicity, speed, portability, and reuse. When building pipelines with Azure Machine Learning, you can focus on your expertise, machine learning, rather than on infrastructure.

Using distinct steps makes it possible to rerun only the steps you need, as you tweak and test your workflow. A step is a computational unit in the pipeline. As shown in the preceding diagram, the task of preparing data can involve many steps. These include, but aren't limited to, normalization, transformation, validation, and featurization. Data sources and intermediate data are reused across the pipeline, which saves compute time and resources.

After the pipeline is designed, there is often more fine-tuning around the training loop of the pipeline. 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 paradigm applies to unchanged scripts used for the execution of the step.

With Azure Machine Learning, you can use various toolkits and frameworks, such as Microsoft Cognitive Toolkit

or TensorFlow, 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.

You can [track the metrics for your pipeline experiments](#) directly in the Azure portal.

Key advantages

The key advantages to building pipelines for your machine learning workflows are:

KEY ADVANTAGE	DESCRIPTION
Unattended runs	Schedule a few steps to run in parallel or in sequence in a reliable and unattended manner. Because 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. You can run individual pipeline steps on different compute targets, such as HDInsight, GPU Data Science VMs, and Databricks. This makes efficient use of available compute options.
Reusability	You can template pipelines for specific scenarios, such as retraining and batch scoring. Trigger them 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. You can also manage scripts and data separately for increased productivity.

The Python SDK for pipelines

Use Python to create your ML pipelines. The Azure Machine Learning SDK offers imperative constructs for sequencing and parallelizing the steps in your pipelines when no data dependency is present. You can interact with it in Jupyter notebooks, or in another preferred integrated development environment.

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. You can extend the framework to model your own conventions, by implementing custom steps that are reusable across pipelines. You can also manage compute targets and storage resources directly from the SDK.

You can save pipelines as templates, and deploy them to a REST endpoint so you can schedule batch-scoring or retraining jobs.

To see how to build your own, see the [Python SDK reference docs for pipelines](#) and the notebook in the next section.

Example notebooks

The following notebooks demonstrate pipelines with Azure Machine Learning: [how-to-use-azureml/machine-learning-pipelines](#).

Learn how to run notebooks by following the article, [Use Jupyter notebooks to explore this service](#).

Next steps

Learn how to [create your first pipeline](#).

Create and manage Azure Machine Learning service workspaces

12/11/2018 • 2 minutes to read • [Edit Online](#)

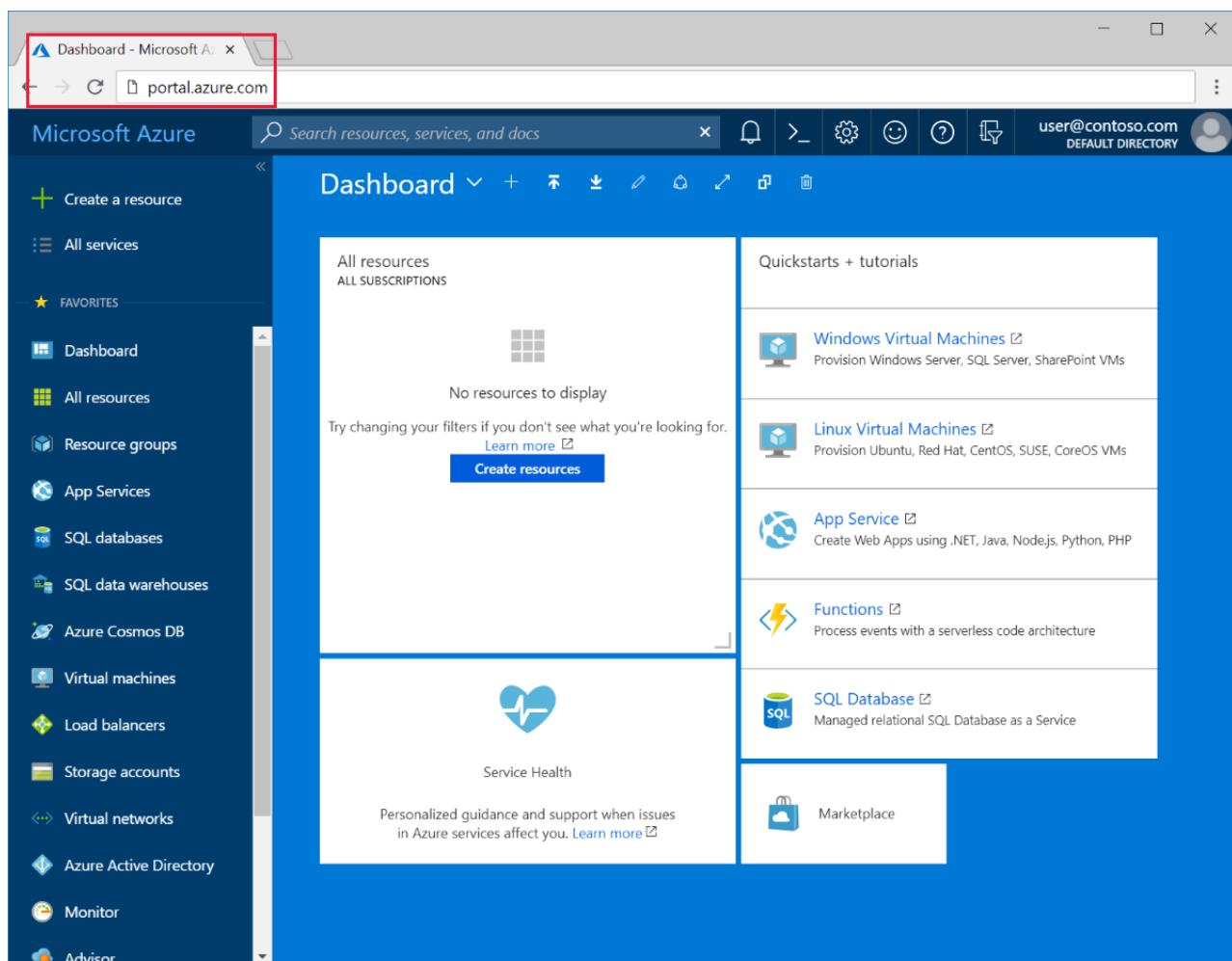
In this article, you'll create, view, and delete **Azure Machine Learning service workspaces** in the Azure portal for [Azure Machine Learning service](#). You can also create and delete workspaces [using the CLI](#) or [with Python code](#).

Create a workspace

To create a workspace, you need an Azure subscription. If you don't have an Azure subscription, create a free account before you begin. Try the [free or paid version of Azure Machine Learning service](#) today.

Sign in to the [Azure portal](#) by using the credentials for the Azure subscription you use.

The portal's workspace dashboard is supported on Edge, Chrome, and Firefox browsers only.



In the upper-left corner of the portal, select **Create a resource**.



In the search bar, enter **Machine Learning**. Select the **Machine Learning service workspace** search result.

Everything

Filter

machine learning service workspace

Results

NAME	PUBLISHER	CATEGORY
Machine Learning service workspace	Microsoft	Analytics
Machine Learning Studio Workspace	Microsoft	Analytics
Machine Learning Studio Web Service Plan	Microsoft	Analytics

In the **Machine Learning service workspace** pane, scroll to the bottom and select **Create** to begin.

Create

In the **ML service workspace** pane, configure your workspace.

FIELD	DESCRIPTION
Workspace name	Enter a unique name that identifies your workspace. Here we use docs-ws. Names must be unique across the resource group. Use a name that's easy to recall and differentiate from workspaces created by others.
Subscription	Select the Azure subscription that you want to use.
Resource group	Use an existing resource group in your subscription, or enter a name to create a new resource group. A resource group is a container that holds related resources for an Azure solution. Here we use docs-am.
Location	Select the location closest to your users and the data resources. This location is where the workspace is created.

ML service workspace

Machine Learning service workspace

* Workspace name
Enter the workspace name

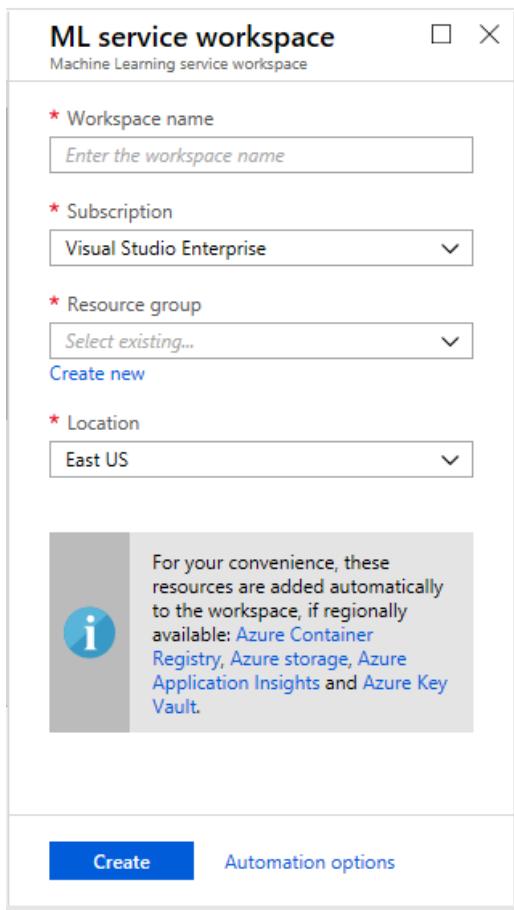
* Subscription
Visual Studio Enterprise

* Resource group
Select existing...
Create new

* Location
East US

i For your convenience, these resources are added automatically to the workspace, if regionally available: Azure Container Registry, Azure storage, Azure Application Insights and Azure Key Vault.

Create Automation options



To start the creation process, select **Create**. It can take a few moments to create the workspace.

To check on the status of the deployment, select the Notifications icon (bell) on the toolbar.



When the process is finished, a deployment success message appears. It's also present in the notifications section.

To view the new workspace, select **Go to resource**.

View a workspace

1. In top left corner of the portal, select **All services**.
2. In the **All services** filter field, type **Machine Learning service workspace**.

All services Filter By category X

GENERAL (14)

- Dashboard
- All resources
- Recent
- Management groups
- Subscriptions
- Resource groups
- Cost Management + Billing PREVIEW
- Reservations
- Marketplace
- Help + support
- Service Health
- Templates PREVIEW
- Tags
- What's new

3. In the filter results, select **Machine Learning service workspace** to display a list of your workspaces.

Everything

Filter

machine learning workspace

Results

NAME	PUBLISHER	CATEGORY
Machine Learning Workspace (preview)	Microsoft	Analytics
Machine Learning Studio Workspace	Microsoft	Analytics
Machine Learning Studio Web Service Plan	Microsoft	Analytics

4. Look through the list of workspaces found. You can filter based on subscription, resource groups, and locations.

Home > Machine Learning Workspaces

Machine Learning Workspaces

Microsoft - PREVIEW

Add Edit columns Refresh Assign tags Delete

Subscriptions: All 30 selected

doc-ws All subscriptions All resource gro... All locations

1 items

NAME	RESOURCE GROUP	LOCATION
doc-ws	docs-aml	East US 2

5. Select the workspace you just created to display its properties.

Home > Machine Learning Workspaces > doc-ws

doc-ws Machine Learning Workspace - PREVIEW

Search (Ctrl+ /)

Overview Activity log Access control (IAM) Tags Diagnose and solve problems

Settings Locks Automation script Properties Application

Delete

Resource group docs-aml Storage docwsxxxxxxxxxx Location East US 2 Registry docwsxxxxxxxxxx Subscription Visual Studio Ultimate with MSDN Key Vault docwsxxxxxxxxxx Subscription ID xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx Application Insights docwsxxxxxxxxxx

Getting Started

Explore your Azure Machine Learning Workspace

Explore your Machine Learning Workspace to run and track experiments, compare model performance, and deploy models.

Delete a workspace

Use the Delete button at the top of the workspace you wish to delete.

Home > Machine Learning Workspaces > doc-ws

doc-ws Machine Learning Workspace - PREVIEW

Search (Ctrl+ /)

Delete

Overview

Resource group docs-aml

Clean up resources

IMPORTANT

The resources you created can be used as prerequisites to other Azure Machine Learning service tutorials and how-to articles.

If you don't plan to use the resources you created here, delete them so you don't incur any charges.

1. In the Azure portal, select **Resource groups** on the far left.

The screenshot shows the Microsoft Azure portal interface. The left sidebar has a red box around the 'Resource groups' option under the 'All resources' section. The main content area is titled 'newacct' and shows the 'Overview' tab selected. At the top right of this area, there is a red box around the 'Delete resource group' button. Below it, the 'Essentials' section displays subscription information and deployment status. A table lists one item: 'newacct' (Azure Cosmos DB account) located in South Central US.

NAME	TYPE	LOCATION
newacct	Azure Cosmos DB account	South Central US

2. From the list, select the resource group you created.
3. Select **Delete resource group**.
4. Enter the resource group name, and then select **Delete**.

Next steps

Follow the full-length tutorial to learn how to use a workspace to build, train, and deploy models with Azure Machine Learning service.

[Tutorial: Train models](#)

Configure a development environment for Azure Machine Learning

12/10/2018 • 9 minutes to read • [Edit Online](#)

In this document, learn how to configure a development environment to work with the Azure Machine Learning service. The Azure Machine Learning service is platform agnostic. The only requirements for your development environment are **Python 3**, **Conda** (for isolated environments), and a configuration file that contains your Azure Machine Learning workspace information.

This document focuses on the following specific environments and tools:

- [Azure Notebooks](#): A Jupyter Notebooks service hosted in the Azure cloud. It is **the easiest** way to get started, as the Azure Machine Learning SDK is already installed.
- [The Data Science Virtual Machine](#): A **pre-configured development/experimentation environment** in the Azure cloud that is **designed for data science work** and can be deployed to either CPU only VM instances or GPU based instances. Python 3, Conda, Jupyter Notebooks, and the Azure Machine Learning SDK are already installed. The VM comes with popular ML / deep learning frameworks, tools, and editors for developing ML solutions. It is probably **the most complete** development environment for ML on the Azure platform.
- [Jupyter Notebooks](#): If you're already using Jupyter Notebooks, the SDK has some extras that you should install.
- [Visual Studio Code](#): If you use Visual Studio Code, there are some useful extensions that you can install.
- [Azure Databricks](#): A popular data analytics platform based on Apache Spark. Learn how to get the Azure Machine Learning SDK onto your cluster so you can deploy models.

If you already have a Python 3 environment, or just want the basic steps for installing the SDK, see the [Local computer](#) section.

Prerequisites

- An Azure Machine Learning service workspace. Follow the steps in [Get started with Azure Machine Learning service](#) to create one.
- Either the [Continuum Anaconda](#) or [Miniconda](#) package manager.

IMPORTANT

Anaconda and Miniconda are not required when using Azure Notebooks.

- On Linux or Mac OS, you need the bash shell.

TIP

If you are on Linux or Mac OS and use a shell other than bash (for example, zsh) you may receive errors when running some commands. To work around this problem, use the `bash` command to start a new bash shell and run the commands there.

- On Windows, you need the command prompt or Anaconda prompt (installed by Anaconda and Miniconda).

Azure Notebooks

[Azure Notebooks](#) (preview) is an interactive development environment in the Azure cloud. It is **the easiest** way to get started with Azure Machine Learning development.

- The Azure Machine Learning SDK is **already installed**.
- After creating an Azure Machine Learning service workspace in the Azure portal, you can click a button to automatically configure your Azure Notebook environment to work with the workspace.

To get started developing with Azure Notebooks, follow the [Get started with Azure Machine Learning service](#) document.

Data Science Virtual Machine

The Data Science Virtual Machine (DSVM) is a customized virtual machine (VM) image **designed for data science work** that is pre-configured with:

- Packages such as Tensorflow, Pytorch, scikit-learn, Xgboost and Azure ML SDK
- Popular data science tools like Spark standalone, Drill
- Azure tools such as the CLI, Azcopy and Storage explorer
- Integrated development environments (IDEs) such as Visual Studio Code, PyCharm and RStudio
- Jupyter Notebook Server

The Azure Machine Learning SDK works on either the Ubuntu or Windows version of DSVM. To use the Data Science Virtual Machine as a development environment, use the following steps:

1. To create a Data Science Virtual Machine, use one of the following methods:

- Using the Azure Portal:
 - [Create an Ubuntu Data Science Virtual Machine](#)
 - [Create a Windows Data Science Virtual Machine](#)
- Using the Azure CLI:

IMPORTANT

When using the Azure CLI, you must first sign in to your Azure subscription by using the `az login` command.

When using the commands in this step, you must provide a resource group name, a name for the VM, a username, and a password.

- To create an **Ubuntu** Data Science Virtual Machine, use the following command:

```
# create a Ubuntu DSVM in your resource group
# note you need to be at least a contributor to the resource group in order to execute
this command successfully
# If you need to create a new resource group use: "az group create --name YOUR-RESOURCE-
GROUP-NAME --location YOUR-REGION (For example: westus2)"
az vm create --resource-group YOUR-RESOURCE-GROUP-NAME --name YOUR-VM-NAME --image
microsoft-dsvm:linux-data-science-vm-ubuntu:linuxdsvmbuntu:latest --admin-username YOUR-
USERNAME --admin-password YOUR-PASSWORD --generate-ssh-keys --authentication-type password
```

- To create a **Windows** Data Science Virtual Machine, use the following command:

```
# create a Windows Server 2016 DSVM in your resource group
# note you need to be at least a contributor to the resource group in order to execute
this command successfully
az vm create --resource-group YOUR-RESOURCE-GROUP-NAME --name YOUR-VM-NAME --image
microsoft-dsvm:dsvm-windows:server-2016:latest --admin-username YOUR-USERNAME --admin-
password YOUR-PASSWORD --authentication-type password
```

2. The Azure Machine Learning SDK is **already installed** on the DSVM. To use the Conda environment that contains the SDK, use one of the following commands:

- On **Ubuntu** DSVM, use this command:

```
conda activate py36
```

- On **Windows** DSVM, use this command:

```
conda activate AzureML
```

3. To verify that you can access the SDK and check the version, use the following Python code:

```
import azureml.core
print(azureml.core.VERSION)
```

4. To configure the DSVM to use your Azure Machine Learning service workspace, see the [Configure workspace](#) section.

For more information on the Data Science Virtual Machines, see [Data Science Virtual Machines](#).

Local computer

When using a local computer (which might also be a remote virtual machine), use the following steps to create a conda environment and install the SDK:

1. Open a command prompt or shell.
2. Create a conda environment with the following commands:

```
# create a new conda environment with Python 3.6, numpy, and cython
conda create -n myenv Python=3.6 cython numpy

# activate the conda environment
conda activate myenv

# On Mac OS run
source activate myenv
```

It might take several minutes to create the environment if Python 3.6 and other components need to be downloaded.

3. Install the Azure Machine Learning SDK with notebook extras and the Data Preparation SDK by using the following command:

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

NOTE

If you get a message that `PyYAML` can't be uninstalled, use the following command instead:

```
pip install --upgrade azureml-sdk[notebooks,automl] azureml-dataprep --ignore-installed PyYAML
```

It might take several minutes to install the SDK.

4. Install packages for your machine learning experimentation. Use the following command and replace `<new package>` with the package you want to install:

```
conda install <new package>
```

5. To verify that the SDK is installed, the following Python code:

```
import azureml.core  
azureml.core.VERSION
```

Jupyter Notebooks

Jupyter Notebooks are part of the [Jupyter Project](#). They provide an interactive coding experience where you create documents that mix live code with narrative text and graphics. Jupyter Notebooks are also a great way to share your results with others, as you can save the output of your code sections in the document. You can install Jupyter Notebooks on a variety of platforms.

The steps in the [Local computer](#) section install optional components for Jupyter Notebooks. To enable these components in your Jupyter Notebook environment, use the following steps:

1. Open a command prompt or shell.
2. To install a conda-aware Jupyter Notebook server using the following command:

```
# install Jupyter  
conda install nb_conda
```

3. Open Jupyter Notebook with the following command:

```
jupyter notebook
```

4. To verify that Jupyter Notebook can use the SDK, open a new notebook and select "myenv" as your kernel. Then run the following command in a notebook cell:

```
import azureml.core  
azureml.core.VERSION
```

5. To configure the Jupyter Notebook to use your Azure Machine Learning service workspace, see the [Configure workspace](#) section.

Visual Studio Code

Visual Studio Code is a cross platform code editor. It relies on a local Python 3 and Conda installation for Python support, but it provides additional tools for working with AI. It also provides support for selecting the Conda environment from within the code editor.

To use Visual Studio Code for development, use the following steps:

- To learn how to use Visual Studio Code for Python development, see the [Get started with Python in VSCode](#) document.
- To select the Conda environment, open VS Code and then use **Ctrl-Shift-P** (Linux and Windows) or **Command-Shift-P** (Mac) to get the **Command Pallet**. Enter **Python: Select Interpreter** and then select the conda environment.
- To validate that you can use the SDK, create a new Python file (.py) that contains the following code. Then run the file:

```
import azureml.core
azureml.core.VERSION
```

- To install the Azure Machine Learning extension for Visual Studio Code, see the [Tools for AI](#) page.

For more information, see [Using Azure Machine Learning for Visual Studio Code](#).

Azure Databricks

You can use a custom version of the Azure Machine Learning SDK for Azure Databricks for end-to-end custom machine learning. Or, train your model within Databricks and use [Visual Studio Code](#) to deploy the model.

To prepare your Databricks cluster and get sample notebooks:

- Create a [Databricks cluster](#) with a Databricks runtime version of 4.x (high concurrency preferred) with [Python 3](#).
- Create a library to [install and attach](#) the Azure Machine Learning SDK for Python `azureml-sdk[databricks]` PyPi package to your cluster. When you are done, you see the library attached as shown in this image. Be aware of these [common Databricks issues](#).

The screenshot shows the Microsoft Azure Databricks interface. On the left, there's a sidebar with icons for Home, Workspace, and Recents. The main area has a header 'Microsoft Azure' and a sub-header 'azureml-sdk[databricks]'. Below that, it says 'azureml-sdk[databricks]' and 'PyPI rules' with a red box around 'azureml-sdk[databricks]'. Under 'Status on running clusters', there's a table:

Attach	Name	Status
<input checked="" type="checkbox"/>	newclus_test	Attached

If this step fails, restart your cluster:

- Select `Clusters` in the left pane. Select your cluster name in the table.
 - On the `Libraries` tab, select `Restart`.
- Download the [Azure Databricks / Azure Machine Learning SDK notebook archive file](#).

WARNING

Many sample notebooks are available for use with Azure Machine Learning service. Only these sample notebooks work with Azure Databricks: <https://github.com/Azure/MachineLearningNotebooks/blob/master/how-to-use-azureml/azure-databricks>

- Import this archive file into your Databricks cluster and start exploring as described here.

Create a workspace configuration file

The workspace configuration file is a JSON document that tells the SDK how to communicate with your Azure Machine Learning service workspace. The file is named `config.json` and it has the following format:

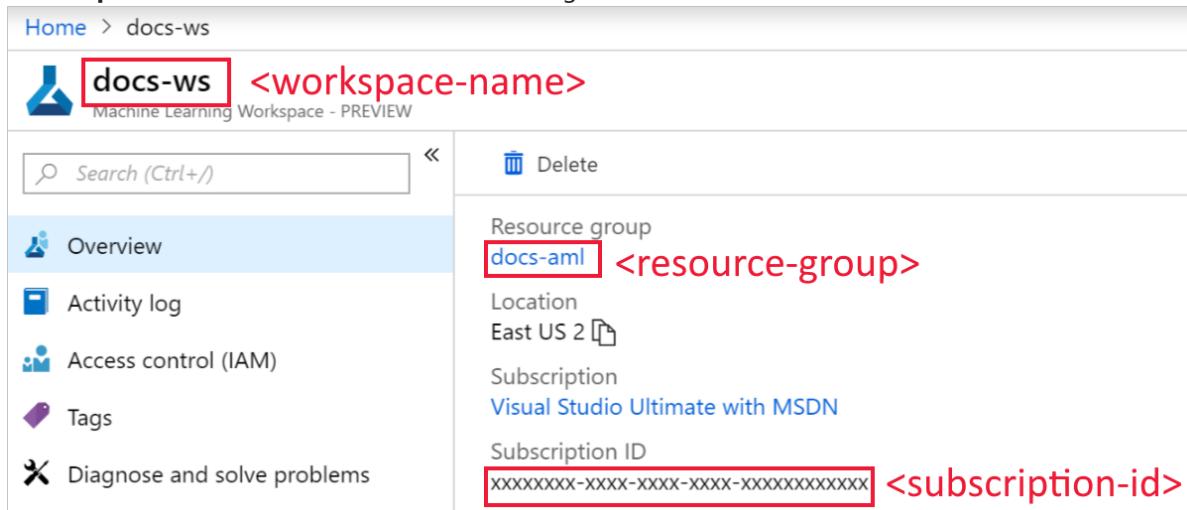
```
{  
    "subscription_id": "<subscription-id>",  
    "resource_group": "<resource-group>",  
    "workspace_name": "<workspace-name>"  
}
```

This file must be in the directory structure that contains your Python scripts or Jupyter Notebooks. It can either be in the same directory, a subdirectory named `aml_config`, or in a parent directory.

To use this file from your code, use `ws=Workspace.from_config()`. This code loads the information from the file and connects to your workspace.

There are three ways to create the configuration file:

- If you follow the [Azure Machine Learning quickstart](#), a `config.json` file is created in your Azure Notebooks library. This file contains the configuration information for your workspace. You can download or copy this `config.json` to other development environments.
- You can **manually create the file** using a text editor. You can find the values for that go into the config file by visiting your workspace in the [Azure portal](#). Copy the **Workspace name**, **Resource group**, and **Subscription ID** values and use them in the configuration file.



- You can **create the file programmatically**. The following code snippet demonstrates how to connect to a workspace by providing the subscription ID, resource group, and workspace name. Then it saves the workspace configuration to file:

```
from azureml.core import Workspace

subscription_id = '<subscription-id>'
resource_group = '<resource-group>'
workspace_name = '<workspace-name>'

try:
    ws = Workspace(subscription_id = subscription_id, resource_group = resource_group, workspace_name =
    workspace_name)
    ws.write_config()
    print('Library configuration succeeded')
except:
    print('Workspace not found')
```

This code writes the configuration file to the `aml_config/config.json` file.

Next steps

- [Train a model on Azure Machine Learning with the MNIST dataset](#)
- [Azure Machine Learning SDK for Python](#)
- [Azure Machine Learning Data Prep SDK](#)

Get started with Azure Machine Learning for Visual Studio Code

12/11/2018 • 4 minutes to read • [Edit Online](#)

In this article, you'll learn how to install the **Azure Machine Learning for Visual Studio Code** extension and create your first experiment with Azure Machine Learning service in Visual Studio Code (VS Code).

Use the Azure Machine Learning extension in Visual Studio code to use the Azure Machine Learning service to prep your data, train, and test machine learning models on local and remote compute targets, deploy those models and track custom metrics and experiments.

Prerequisite

- If you don't have an Azure subscription, create a free account before you begin. Try the [free or paid version of Azure Machine Learning service](#) today.
- Visual Studio Code must be installed. VS Code is a lightweight but powerful source code editor that runs on your desktop. It comes with built-in support for Python and more. [Learn how to install VS Code](#).
- [Install Python 3.5 or greater](#).

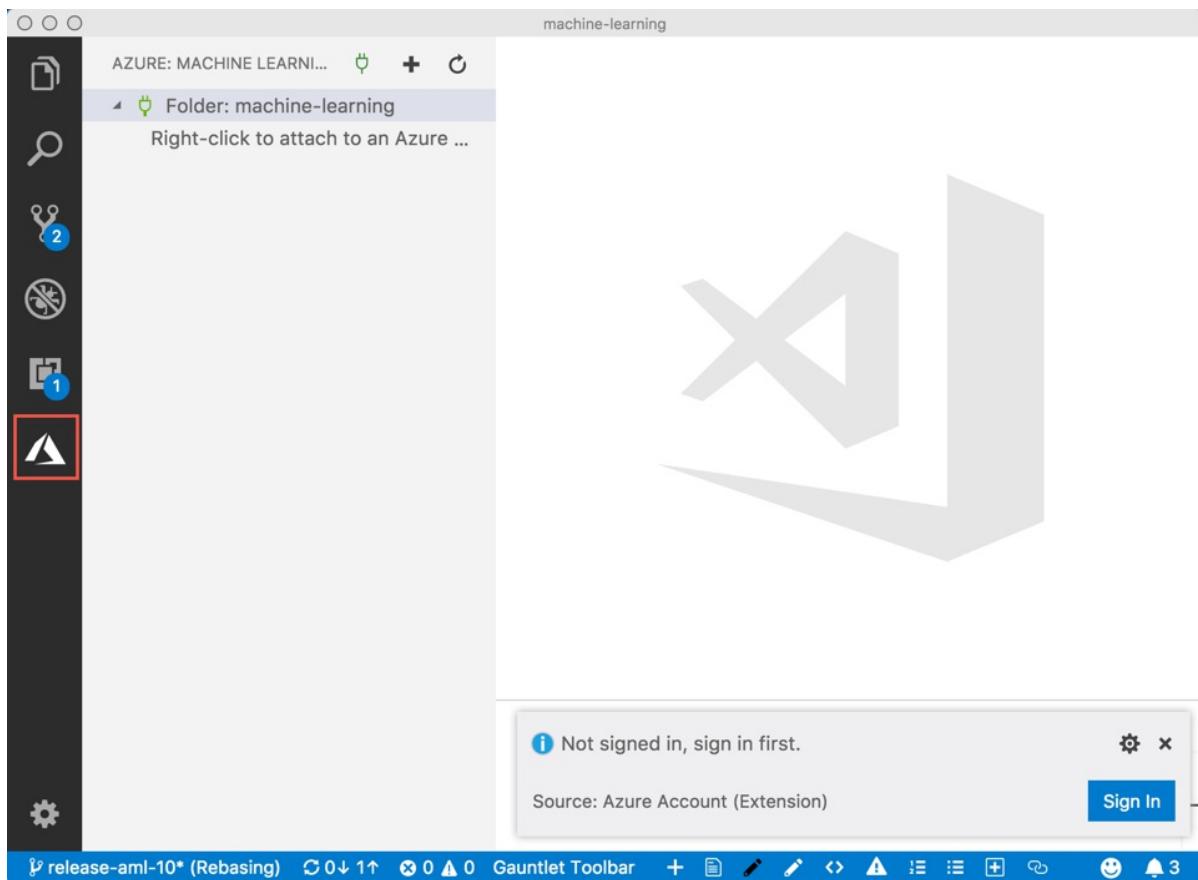
Install the Azure Machine Learning for VS Code extension

When you install the **Azure Machine Learning** extension, two more extensions are automatically installed (if you have internet access). They are the [Azure Account](#) extension and the [Microsoft Python](#) extension

To work with Azure Machine Learning, we need to turn VS Code into a Python IDE. Working with [Python in Visual Studio Code](#), requires the Microsoft Python extension, which gets installed with the Azure Machine Learning extension automatically. The extension makes VS Code an excellent IDE, and works on any operating system with a variety of Python interpreters. It leverages all of VS Code's power to provide auto complete and IntelliSense, linting, debugging, and unit testing, along with the ability to easily switch between Python environments, including virtual and conda environments. Check out this walk-through of editing, running, and debugging Python code, see the [Python Hello World Tutorial](#)

To install the Azure Machine Learning extension:

1. Launch VS Code.
2. In a browser, visit: [Azure Machine Learning for Visual Studio Code \(Preview\)](#) extension
3. In that web page, click **Install**.
4. In the extension tab, click **Install**.
5. A welcome tab opens in VS Code for the extension and the Azure symbol is added to activity bar.



6. In the dialog box, click **Sign In** and follow the onscreen prompt to authenticate with Azure.

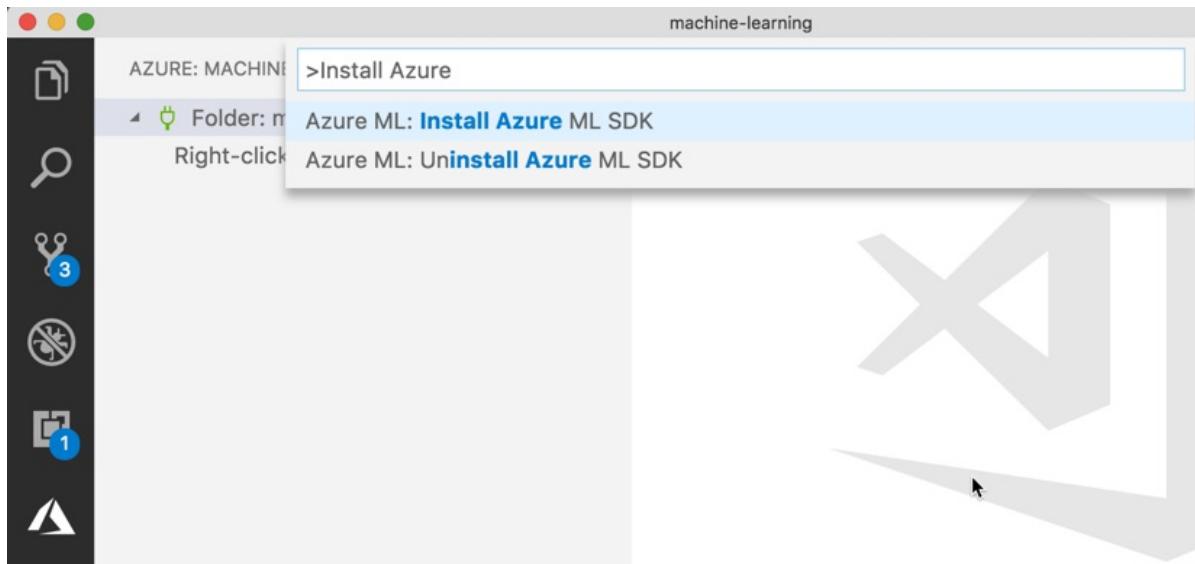
The Azure Account extension, which was installed along with the Azure Machine Learning for VS Code extension, helps you authenticate with your Azure account. See the list of commands in the [Azure Account extension](#) page.

TIP

Check out the [IntelliCode extension for VS Code \(preview\)](#). IntelliCode provides a set of AI-assisted capabilities for IntelliSense in Python, such as inferring the most relevant auto-completions based on the current code context.

Install the SDK

1. Make sure that Python 3.5 or greater is installed and recognized by VS Code. If you install it now, then restart VS Code and select a Python interpreter using instructions at <https://code.visualstudio.com/docs/python/python-tutorial>.
2. In VS Code, open the Command Palette **Ctrl+Shift+P**.
3. Type 'Install Azure ML SDK' to find the pip install command for the SDK. A local private Python environment is created that has the Visual Studio Code prerequisites for working with Azure Machine Learning.



4. In the integrated terminal window, specify the Python interpreter to use or you can hit **Enter** to use your default Python interpreter.

```
PROBLEMS (22) OUTPUT DEBUG CONSOLE TERMINAL 1: Azure ML Package + -x 🗑️ ⌂ ⌄ ⌁ ⌂ ⌁ ×
bash-3.2$ bash setup_aml_env.sh 'python'
Enter path to Python3 interpreter [default: python]:
```

Get started with Azure Machine Learning

Before you start training and deploying machine learning models using VS Code, you need to create an [Azure Machine Learning service workspace](#) in the cloud to contain your models and resources. Learn how to create one and create your first experiment in that workspace.

1. Click the Azure icon in the Visual Studio Code activity bar. The Azure Machine Learning sidebar appears.

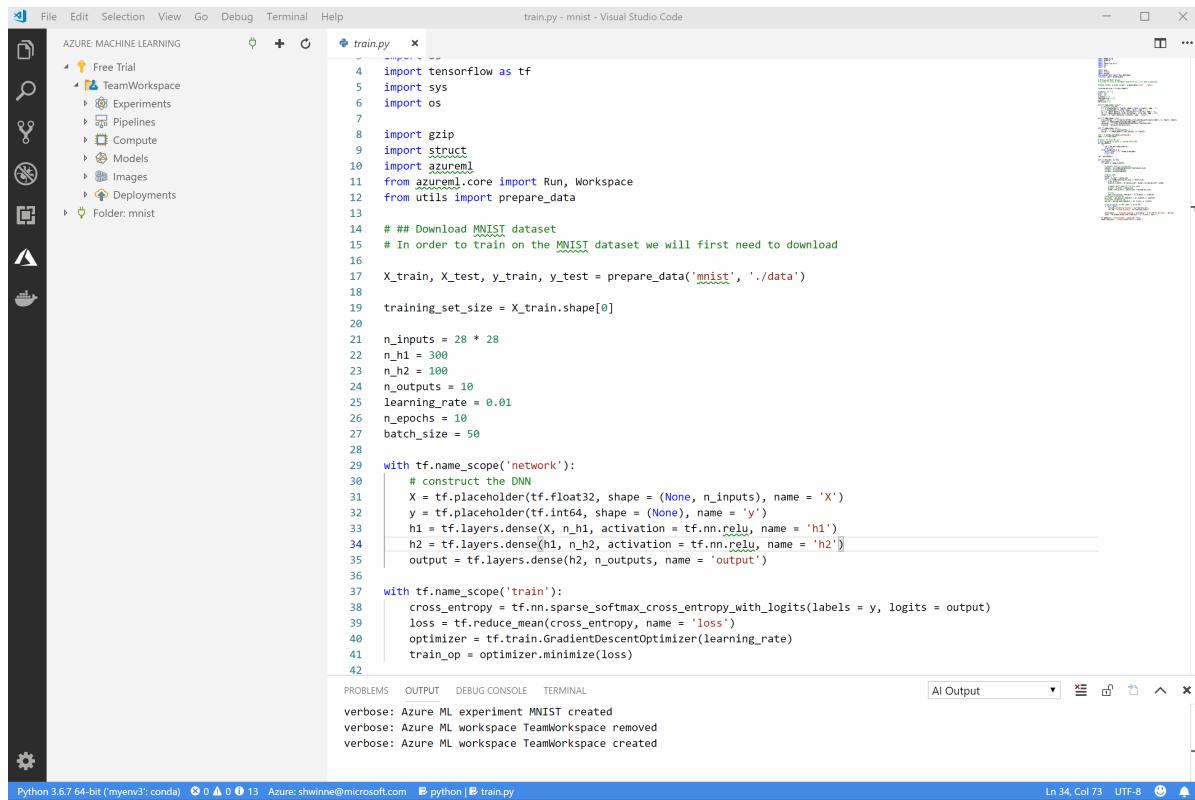
```
File Edit Selection View Go Debug Terminal Help train.py - mnist - Visual Studio Code
AZURE: MACHINE LEARNING
Free Trial
Click here to create a workspace
Folder: mnist

train.py
1 #!/usr/bin/env python
2
3 import tensorflow as tf
4 import sys
5 import os
6
7 import gzip
8 import struct
9 import azureml
10 from azureml.core import Run, Workspace
11 from utils import prepare_data
12
13
14 # ## Download MNIST dataset
15 # In order to train on the MNIST dataset we will first need to download
16
17 X_train, X_test, y_train, y_test = prepare_data('mnist', './data')
18
19 training_set_size = X_train.shape[0]
20
21 n_inputs = 28 * 28
22 n_h1 = 300
23 n_h2 = 100
24 n_outputs = 10
25 learning_rate = 0.01
26 n_epochs = 10
27 batch_size = 50
28
29 with tf.name_scope('network'):
30     # construct the DNN
31     X = tf.placeholder(tf.float32, shape = (None, n_inputs), name = 'X')
32     y = tf.placeholder(tf.int64, shape = (None), name = 'y')
33     h1 = tf.layers.dense(X, n_h1, activation = tf.nn.relu, name = 'h1')
34     h2 = tf.layers.dense(h1, n_h2, activation = tf.nn.relu, name = 'h2')
35     output = tf.layers.dense(h2, n_outputs, name = 'output')
36
37 with tf.name_scope('train'):
38     cross_entropy = tf.nn.sparse_softmax_cross_entropy_with_logits(labels = y, logits = output)
39     loss = tf.reduce_mean(cross_entropy, name = 'loss')
40     optimizer = tf.train.GradientDescentOptimizer(learning_rate)
41     train_op = optimizer.minimize(loss)
42

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL AI Output
verbose: Azure ML experiment MNIST created
verbose: Azure ML workspace Teamworkspace removed
Python 3.6.7 64-bit ('myenv3:conda') 0 0 0 0 13 Azure: shwinne@microsoft.com python | train.py
Ln 16, Col 1 UTF-8
```

2. Right-click your Azure subscription and select **Create Workspace**. A list appears. In the animated image, the subscription name is 'Free Trial' and the workspace is 'TeamWorkspace'.
3. Select an existing resource group from the list or create a new one using the wizard in the Command Palette.
4. In the field, type a unique and clear name for your new workspace. In the screenshots, the workspace is named 'TeamWorkspace'.
5. Hit enter and the new workspace is created. It appears in the tree below the subscription name.
6. Right-click on the Experiment node and choose **Create Experiment** from the context menu. Experiments keep track of your runs using Azure Machine Learning.
7. In the field, enter a name your experiment. In the screenshots, the experiment is named 'MNIST'.
8. Hit enter and the new experiment is created. It appears in the tree below the workspace name.
9. Right-click the experiment name and choose **Attach Folder to Experiment**. This folder should contain your local Python scripts. The folder is then linked to the experiment in the cloud.

Now each of your experiment runs with your experiment so all of your key metrics will be stored in the experiment history and the models you train will get automatically uploaded to Azure Machine Learning and stored with your experiment metrics and logs.



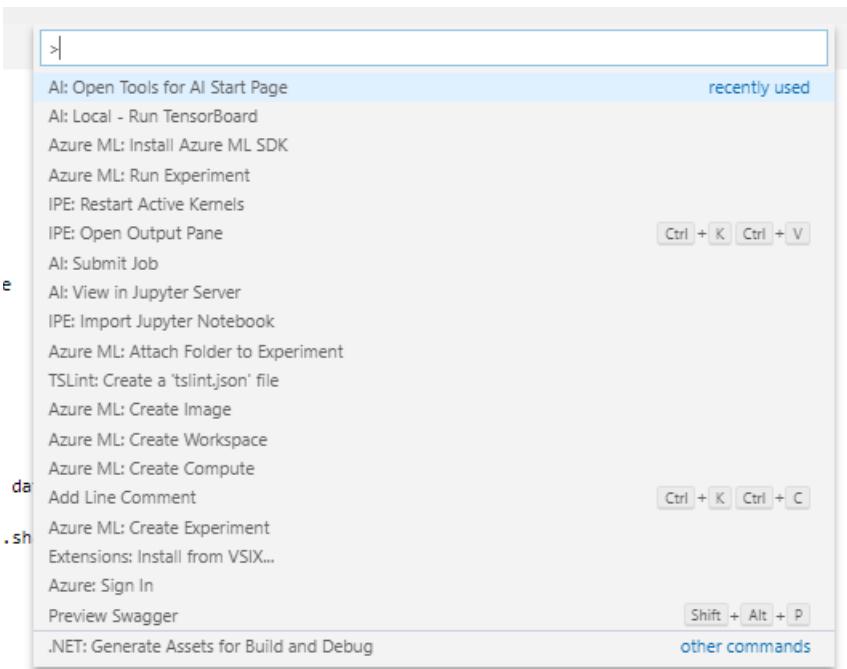
```

AZURE MACHINE LEARNING
File Edit Selection View Go Debug Terminal Help
train.py - mnist - Visual Studio Code
train.py x
1 #!/usr/bin/env python
2
3 import tensorflow as tf
4 import sys
5 import os
6
7
8 import gzip
9 import struct
10 import azureml
11 from azureml.core import Run, Workspace
12 from utils import prepare_data
13
14 # ## Download MNIST dataset
15 # In order to train on the MNIST dataset we will first need to download
16
17 X_train, X_test, y_train, y_test = prepare_data('mnist', './data')
18
19 training_set_size = X_train.shape[0]
20
21 n_inputs = 28 * 28
22 n_h1 = 300
23 n_h2 = 100
24 n_outputs = 10
25 learning_rate = 0.01
26 n_epochs = 10
27 batch_size = 50
28
29 with tf.name_scope('network'):
30     # construct the DNN
31     X = tf.placeholder(tf.float32, shape = (None, n_inputs), name = 'X')
32     y = tf.placeholder(tf.int64, shape = (None), name = 'y')
33     h1 = tf.layers.dense(X, n_h1, activation = tf.nn.relu, name = 'h1')
34     h2 = tf.layers.dense(h1, n_h2, activation = tf.nn.relu, name = 'h2')
35     output = tf.layers.dense(h2, n_outputs, name = 'output')
36
37 with tf.name_scope('train'):
38     cross_entropy = tf.nn.sparse_softmax_cross_entropy_with_logits(labels = y, logits = output)
39     loss = tf.reduce_mean(cross_entropy, name = 'loss')
40     optimizer = tf.train.GradientDescentOptimizer(learning_rate)
41     train_op = optimizer.minimize(loss)
42
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL
verbose: Azure ML experiment MNIST created
verbose: Azure ML workspace TeamWorkspace removed
verbose: Azure ML workspace TeamWorkspace created

```

Use keyboard shortcuts

Like most of VS Code, the Azure Machine Learning features in VS Code are accessible from the keyboard. The most important key combination to know is **Ctrl+Shift+P**, which brings up the Command Palette. From here, you have access to all of the functionality of VS Code, including keyboard shortcuts for the most common operations.



Next steps

You can now use Visual Studio Code to work with Azure Machine Learning.

Learn how to [create compute targets, train, and deploy models in Visual Studio Code](#).

Use Visual Studio Code to train and deploy machine learning models

12/11/2018 • 5 minutes to read • [Edit Online](#)

In this article, you will learn how to use the **Azure Machine Learning for Visual Studio Code** extension to train and deploy machine learning and deep learning models with Azure Machine Learning service in Visual Studio Code (VS Code).

Azure Machine Learning provides support for running experiments locally and on remote compute targets. For every experiment, you can keep track of multiple runs as often you need to iteratively try different techniques, hyperparameters, and more. You can use Azure Machine Learning to track custom metrics and experiment runs, enabling data science reproducibility and auditability.

And you can deploy these models for your testing and production needs.

Prerequisites

- If you don't have an Azure subscription, create a free account before you begin. Try the [free or paid version of Azure Machine Learning service](#) today.
- Have the [Azure Machine Learning for VS Code](#) extension set up.
- Have the [Azure Machine Learning SDK for Python installed](#) with VS Code.

Create and manage compute targets

With Azure Machine Learning for VS Code, you can prepare your data, train models, and deploy them both locally and on remote compute targets.

This extension supports several different remote compute targets for Azure Machine Learning. See the [full list of supported compute targets](#) for Azure Machine Learning.

Create compute targets for Azure Machine Learning in VS Code

To create a compute target:

1. Click the Azure icon in the Visual Studio Code activity bar. The Azure: Machine Learning sidebar appears.
2. In the tree view, expand your Azure subscription and Azure Machine Learning service workspace. In the animated image, the subscription name is 'Free Trial' and the workspace is 'TeamWorkspace'.
3. Under the workspace node, right-click the **Compute** node and choose **Create Compute**.
4. Choose the compute target type from the list.
5. Specify any advanced properties in the JSON config file that opens in a new tab. You can specify a unique name for the compute target in this file.
6. When you are done configuring your compute target, click **Submit** in the lower right.

Here is an example for Azure Machine Learning Compute (AMLCompute):

```

AZURE: MACHINE LEARNING
File Edit Selection View Go Debug Terminal Help
+ train.py x
1 import numpy as np
2 import argparse
3 import os
4 import tensorflow as tf
5 import sys
6 import os
7
8 import gzip
9 import struct
10 import azureml
11 from azureml.core import Run, Workspace
12 from utils import prepare_data
13
14 # ## Download MNIST dataset
15 # In order to train on the MNIST dataset we will first need to download
16
17 X_train, X_test, y_train, y_test = prepare_data('mnist', './data')
18
19 training_set_size = X_train.shape[0]
20
21 n_inputs = 28 * 28
22 n_h1 = 300
23 n_h2 = 100
24 n_outputs = 10
25 learning_rate = 0.01
26 n_epochs = 10
27 batch_size = 50
28
29 with tf.name_scope('network'):
30     # construct the DNN
31     X = tf.placeholder(tf.float32, shape = (None, n_inputs), name = 'X')
32     y = tf.placeholder(tf.int64, shape = (None), name = 'y')
33     h1 = tf.layers.dense(X, n_h1, activation = tf.nn.relu, name = 'h1')
34     h2 = tf.layers.dense(h1, n_h2, activation = tf.nn.relu, name = 'h2')
35     output = tf.layers.dense(h2, n_outputs, name = 'output')
36
37 with tf.name_scope('train'):
38     cross_entropy = tf.nn.sparse_softmax_cross_entropy_with_logits(labels = y, logits = output)
39     loss = tf.reduce_mean(cross_entropy, name = 'loss')
40     optimizer = tf.train.GradientDescentOptimizer(learning_rate)
41     train_op = optimizer.minimize(loss)
42
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL
verbose: Azure ML Compute mycluster removed
verbose: Editing azureml_compute_AmlCompute_create.json finished.
verbose: Azure ML Compute undefined created
verbose: Azure ML Compute mycluster removed
Ln 1, Col 1 Spaces: 4 UTF-8 CR/LF Python

```

Use remote computes for experiments in VS Code

To use a remote compute target when training, you need to create a run configuration file. This file tells Azure Machine Learning not only where to run your experiment but also how to prepare the environment.

The 'run configuration' file

The VS Code extension will automatically create a run configuration for your **local** and **docker** environments on your local computer.

This is a snippet from the default run configuration file.

If you want to install all of your libraries/dependencies yourself, set `userManagedDependencies: True` and then local experiment runs will use your default Python environment as specified by the VS Code Python extension.

```

# user_managed_dependencies=True indicates that the environment will be user managed. False indicates that
# AzureML will manage the user environment.
userManagedDependencies: False
# The python interpreter path
interpreterPath: python
# Path to the conda dependencies file to use for this run. If a project
# contains multiple programs with different sets of dependencies, it may be
# convenient to manage those environments with separate files.
condaDependenciesFile: aml_config/conda_dependencies.yml
# Docker details
docker:
# Set True to perform this run inside a Docker container.
enabled: false

```

The conda dependencies file

By default, a new conda environment is created for you and your installation dependencies are managed. However, you must specify your dependencies in the `aml_config/conda_dependencies.yml` file.

This is a snippet from the default `'aml_config/conda_dependencies.yml'`. You can add additional dependencies in the config file.

```
# The dependencies defined in this file will be automatically provisioned for runs with
userManagedDependencies=False.

name: project_environment
dependencies:
    # The python interpreter version.

    # Currently Azure ML only supports 3.5.2 and later.

    - python=3.6.2

    - pip:
        # Required packages for AzureML execution, history, and data preparation.

        - --index-url https://azuresdktestpypi.azureedge.net/sdk-release/Preview/E7501C02541B433786111FE8E140CAA1
        - --extra-index-url https://pypi.python.org/simple
        - azureml-defaults
```

Train and tune models

Use Azure Machine Learning for VS Code (Preview) to rapidly iterate on your code, step through and debug, and use your source code control solution of choice.

To run your experiment with Azure Machine Learning:

1. Click the Azure icon in the Visual Studio Code activity bar. The Azure: Machine Learning sidebar appears.
2. In the tree view, expand your Azure subscription and Azure Machine Learning service workspace.
3. Under the workspace node, expand the **Compute** node and right-click the **Run Config** of compute you want to use.
4. Select **Run Experiment**.
5. Click **View Experiment Run** to see the integrated Azure Machine Learning portal to monitor your runs and see your trained models.

Deploy and manage models

Azure Machine Learning enables deploying and managing your machine learning models in the cloud and on the edge.

Register your model to Azure Machine Learning from VS Code

Now that you have trained your model, you can register it in your workspace. Registered models can be tracked and deployed.

To register your model:

1. Click the Azure icon in the Visual Studio Code activity bar. The Azure: Machine Learning sidebar appears.
2. In the tree view, expand your Azure subscription and Azure Machine Learning service workspace.
3. Under the workspace node, right-click **Models** and choose **Register Model**.
4. From the list, choose whether you want to upload a **model file** (for single models) a **model folder** (for models with multiple files, such as Tensorflow).
5. Select your folder or file.
6. When you are done configuring your model properties, click **Submit** in the lower right.

Deploy your service from VS Code

Using VS Code, you can deploy your web service to:

- Azure Container Instance (ACI): for testing
- Azure Kubernetes Service (AKS): for production

You do not need to create an ACI container to test in advance since they are created on the fly. However, AKS clusters do need to be configured in advance.

Learn more about [deployment with Azure Machine Learning](#) in general.

To deploy a web service:

1. Click the Azure icon in the Visual Studio Code activity bar. The Azure: Machine Learning sidebar appears.
2. In the tree view, expand your Azure subscription and your Azure Machine Learning service workspace.
3. Under the workspace node, expand the **Models** node.
4. Right-click the model you want to deploy and choose **Deploy Service from Registered Model** command from the context menu.
5. In the Command Palette, choose the compute target to which to deploy from the list.
6. In the field, enter a name for this service.
7. In the Command Palette, press the Enter key on your keyboard to browse and select the script file.
8. In the Command Palette, press the Enter key on your keyboard to browse and select the conda dependency file.
9. When you are done configuring your service properties, click **Submit** in the lower right. In this service properties file, you can specify a local Docker file or a schema.json file that you may want to use.

The web service is now deployed.

Next steps

For a walk-through of training with Machine Learning outside of VS Code, read [Tutorial: Train models with Azure Machine Learning](#).

For a walk-through of editing, running, and debugging code locally, see the [Python Hello World Tutorial](#)

Load and read data with Azure Machine Learning

12/10/2018 • 7 minutes to read • [Edit Online](#)

In this article, you learn different methods of loading data using the [Azure Machine Learning Data Prep SDK](#). The SDK supports multiple data ingestion features, including:

- 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

Load text line data

To read simple text data into a dataflow, use the `read_lines()` without specifying optional parameters.

```
dataflow = dprep.read_lines(path='./data/text_lines.txt')
dataflow.head(5)
```

	LINE
0	Date Minimum temperature Maximum temperature
1	2015-07-1 -4.1 10.0
2	2015-07-2 -0.8 10.8
3	2015-07-3 -7.0 10.5
4	2015-07-4 -5.5 9.3

After the data is ingested, run the following code to convert the dataflow object into a Pandas dataframe.

```
pandas_df = dataflow.to_pandas_dataframe()
```

Load CSV data

When reading delimited files, the underlying runtime can infer the parsing parameters (separator, encoding, whether to use headers, etc.). Run the following code to attempt to read a CSV file by specifying only its location.

```
# SAS expires June 16th, 2019
dataflow =
dprep.read_csv(path='https://dpreptestfiles.blob.core.windows.net/testfiles/read_csv_duplicate_headers.csv?
st=2018-06-15T23%3A01%3A42Z&se=2019-06-16T23%3A01%3A00Z&sp=r&sv=2017-04-
17&r=b&sig=ugQQCmeC2eBamm6ynM7wnI%2BI3TTDTM6z9RPKj4a%2FU6g%3D')
dataflow.head(5)
```

	STNAM	FIPST	LEAID	LEANM10	NCESSCH	MAM_MTH00NUMVALID_1011
0		stnam	fipst	leaid	leanm10	ncessch
1	ALABAMA	1	101710	Hale County	10171002158	
2	ALABAMA	1	101710	Hale County	10171002162	
3	ALABAMA	1	101710	Hale County	10171002156	
4	ALABAMA	1	101710	Hale County	10171000588	2

To exclude lines during loading, define the `skip_rows` parameter. This parameter will skip loading rows descending in the CSV file (using a one-based index).

```
dataflow =
dprep.read_csv(path='https://dpreptestfiles.blob.core.windows.net/testfiles/read_csv_duplicate_headers.csv',
                skip_rows=1)
dataflow.head(5)
```

	STNAM	FIPST	LEAID	LEANM10	NCESSCH	MAM_MTH00NUMVALID_1011
0	ALABAMA	1	101710	Hale County	10171002158	29
1	ALABAMA	1	101710	Hale County	10171002162	40
2	ALABAMA	1	101710	Hale County	10171002156	43
3	ALABAMA	1	101710	Hale County	10171000588	2
4	ALABAMA	1	101710	Hale County	10171000589	23

Run the following code to display the column data types.

```
dataflow.head(1).dtypes

stnam          object
fipst          object
leaid          object
leanm10        object
ncessch        object
schnam10       object
MAM_MTH00numvalid_1011    object
dtype: object
```

By default, the Azure Machine Learning Data Prep SDK does not change your data type. The data source you're reading from is a text file, so the SDK reads all values as strings. For this example, numeric columns should be parsed as numbers. Set the `inference_arguments` parameter to `InferenceArguments.current_culture()` to automatically infer and convert the column types during the file read.

```

dataflow =
dprep.read_csv(path='https://dpreptestfiles.blob.core.windows.net/testfiles/read_csv_duplicate_headers.csv',
                skip_rows=1,
                inference_arguments=dprep.InferenceArguments.current_culture())
dataflow.head(1).dtypes

stnam          object
fipst         float64
leaid         float64
leanm10        object
ncessch       float64
schnam10        object
ALL_MTH00numvalid_1011   float64
dtype: object

```

Several of the columns were correctly detected as numeric and their type is set to `float64`.

Use Excel data

The SDK includes a `read_excel()` function to load Excel files. By default the function will load the first sheet in the workbook. To define a specific sheet to load, define the `sheet_name` parameter with the string value of the sheet name.

```

dataflow = dprep.read_excel(path='./data/excel.xlsx', sheet_name='Sheet2')
dataflow.head(5)

```

	COLUMN1	COLUMN2	COLUMN3	COLUMN4	COLUMN5	COLUMN6	COLUMN7	COLUMN8
0	None	None	None	None	None	None	None	None
1	None	None	None	None	None	None	None	None
2	None	None	None	None	None	None	None	None
3	Rank	Title	Studio	Worldwide	Domestic / %	Column1	Overseas / %	Column2
4	1	Avatar	Fox	2788	760.5	0.273	2027.5	0.727

The output shows that the data in the second sheet had three empty rows before the headers. The `read_excel()` function contains optional parameters for skipping rows and using headers. Run the following code to skip the first three rows, and use the fourth row as the headers.

```

dataflow = dprep.read_excel(path='./data/excel.xlsx', sheet_name='Sheet2', use_header=True, skip_rows=3)

```

	RANK	TITLE	STUDIO	WORLD WIDE	DOMESTIC / %	COLUMN 1	OVERSEAS / %	COLUMN 2	YEAR^
0	1	Avatar	Fox	2788	760.5	0.273	2027.5	0.727	2009^
1	2	Titanic	Par.	2186.8	658.7	0.301	1528.1	0.699	1997^

	RANK	TITLE	STUDIO	WORLD WIDE	DOMESTIC / %	COLUMN 1	OVERSEAS / %	COLUMN 2	YEAR^
2	3	Marvel's The Avengers	BV	1518.6	623.4	0.41	895.2	0.59	2012
3	4	Harry Potter and the Deathly Hallows Part 2	WB	1341.5	381	0.284	960.5	0.716	2011
4	5	Frozen	BV	1274.2	400.7	0.314	873.5	0.686	2013

Load fixed-width data files

To load fixed-width files, you specify a list of character offsets. The first column is always assumed to start at zero offset.

```
dataflow = dprep.read_fwf('./data/fixed_width_file.txt', offsets=[7, 13, 43, 46, 52, 58, 65, 73])
dataflow.head(5)
```

	010000	99999	BOGUS NORWAY	NO	NO_1	ENRS	COLUMN 7	COLUMN 8	COLUMN 9
0	010003	99999	BOGUS NORWA Y	NO	NO	ENSO			
1	010010	99999	JAN MAYEN	NO	JN	ENJA	+70933	-008667	+00090
2	010013	99999	ROST	NO	NO				
3	010014	99999	SOERST OKKEN	NO	NO	ENSO	+59783	+00535 0	+00500
4	010015	99999	BRINGEL AND	NO	NO	ENBL	+61383	+00586 7	+03270

To avoid header detection and parse the correct data, pass `PromoteHeadersMode.NONE` to the `header` parameter.

```
dataflow = dprep.read_fwf('./data/fixed_width_file.txt',
                           offsets=[7, 13, 43, 46, 52, 58, 65, 73],
                           header=dprep.PromoteHeadersMode.NONE)
```

	COLUMN 1	COLUMN 2	COLUMN 3	COLUMN 4	COLUMN 5	COLUMN 6	COLUMN 7	COLUMN 8	COLUMN 9
0	010000	99999	BOGUS NORWA Y	NO	NO_1	ENRS	Column 7	Column 8	Column 9

	COLUMN 1	COLUMN 2	COLUMN 3	COLUMN 4	COLUMN 5	COLUMN 6	COLUMN 7	COLUMN 8	COLUMN 9
1	010003	99999	BOGUS NORWAY	NO	NO	ENSO			
2	010010	99999	JAN MAYEN	NO	JN	ENJA	+70933	-008667	+00090
3	010013	99999	ROST	NO	NO				
4	010014	99999	SOERSTOKKEN	NO	NO	ENSO	+59783	+005350	+00500
5	010015	99999	BRINGELAND	NO	NO	ENBL	+61383	+005867	+03270

Load SQL data

The SDK can also load data from a SQL source. Currently, only Microsoft SQL Server is supported. To read data from a SQL server, create a `MSSQLDataSource` object that contains the connection parameters. The `password` parameter of `MSSQLDataSource` accepts a `Secret` object. You can build a secret object in two ways:

- Register the secret and its value with the execution engine.
 - Create the secret with only an `id` (if the secret value is already registered in the execution environment) using `dprep.create_secret("[SECRET-ID]")`.

```
secret = dprep.register_secret(value="[SECRET-PASSWORD]", id="[SECRET-ID]")

ds = dprep.MSSQLDataSource(server_name="[SERVER-NAME]",
                           database_name="[DATABASE-NAME]",
                           user_name="[DATABASE-USERNAME]",
                           password=secret)
```

After you create a data source object, you can proceed to read data from query output.

```
dataflow = dprep.read_sql(ds, "SELECT top 100 * FROM [SalesLT].[Product]")
dataflow.head(5)
```

		PR	O	D	UC	ST	AN	DA	LIS	WE	OR	OD	PR	SE	DI	TH	TH	U	M	BN	AIL	PH	OT	RO	M
PR	O	UC	TN	AN	DA	LIS	WE	OR	YI	ELI	D	TC	OD	LL	SE	ON	M	TI	BN	OF	AIL	PH	NA	GU	ED
O	D	U	CO	RD	TP	SIZ	IG	YI	ELI	D	D	EG	TM	AR	LLE	NU	AIL	ED	PH	OT	NA	ME	ID	TE	DA
D	UC	M	CO	RD	TP	SIZ	E	HT	D	D	D	OR	OD	TD	ND	DA	DA	OT	NA	GU	ED	DA	TE	ME	TE
UC	TI	NA	BE	LO	CO	RI	CE	E	HT	D	D	YI	ELI	AT	DA	DA	O	ME	ID	TE	DA	OT	NA	GU	DA
TI	D	ME	R	R	ST	CE	E	HT	D	D	D	IG	ELI	AT	TE	TE	O	ME	ID	TE	DA	OT	NA	GU	DA

		PR O D UC TI D	PR O D UC TN U M M BE R	PR O D UC TN U M M BE R	ST AN DA	LIS		WE IG HT	PR O D UC TC AT EG	PR O D UC TM OD	SE LL ST AR	DI SC ON TI NU ED	TH U M BN AIL	TH U M BN AIL	TH U M BN AIL	RO W GU ID	M OD IFI ED DA TE	
		NA ME	CO LO R	RD CO ST	TP RI CE	SIZ E		WE IG HT	YI D	ELI D	TD AT E	DA TE	DA TE	DA TE	DA TE	DA TE	DA TE	
0	68	HL	FR	Bl	10	14	58	10	18	6	20	No	No	b'	no	43	20	
0	Ro ad	-	ac	59	31			16		02	-	ne	ne	Gl	_i	dd	08	
	R9	R9	k	.3	.5			.0			06			F8	m	68	-	
	Fr	2B		10	0			4			-			9a	ag	d6	03	
	a	-		0							01			P\	e_	-	-	
	m			58							00			x0	av	14	11	
	e	-									:0			01	ail	a4		
	Bl										0:			\x	abl	-		
	ac										00			00	e_	46		
	k,										+0			\xf	sm	1f-		
	58										0:			7\	all.	90		
											00			x0	gif	69		
														0\	-			
														x0	55			
														0\	30			
														x0	9d			
														0\	90			
														x0	ea			
														0\	7e			
														x0				
														0\				
														x8				
														0...				
1	70	HL	FR	Re	10	14	58	10	18	6	20	No	No	b'	no	95	20	
6	Ro ad	-	rd	59	31			16		02	-	ne	ne	Gl	_i	40	08	
	R9	R9		.3	.5			.0			06			F8	m	ff1	-	
	Fr	2R		10	0			4			-			9a	ag	7-	03	
	a	-		0							01			P\	e_	27	-	
	m	-		58							00			x0	av	12	11	
	e	-									:0			01	ail	-		
	Re										0:			\x	abl	4c		
	d										00			00	e_	90		
	58										+0			\xf	sm	-		
											0:			7\	all.	a3		
											00			x0	gif	d1		
														0\	-			
														x0	8c			
														0\	e5			
														x0	56			
														0\	8b			
														x0	24			
														0\	62			
														x0				
														0\				
														x8				
														0...				

		PR O D UC TI D	NA ME	PR O D UC TN U M BE R	UC TN U M BE R	ST AN DA RD CO ST	LIS TP RI CE	SIZ E	WE IG HT	PR OD UC TC AT EG OR YI D	OD UC TM OD AT ELI D	PR OD UC TM AR TD AT E	SE LL ST AR ND DA TE	DI SC ON TI NU ED DA TE	TH U M BN AI PH OT OF ILE NA ME	TH U M BN AI PH OT OF ILE NA ME	M OD IFI ED DA TE		
4	70 9	Mount ain - Bike So ck s, M	S O- 9 - B9 M	W hit e	3. 39 63	9. 50	M	No ne	27	18	20 05 - 07 - 01 00 :0 0: 00 +0 0: 00	20 06 - 06 - 30 00 :0 0: 00 +0 0: 00	No ne	b' Gl F8 9a P\ x0 01 01 \x 00 \f 7\ x0 0\ x0 0\ x0 0\ x0 0\ x8 0...	no _i m ag e_ av ail 40 abl sm all. gif	18 f9 5f 47 - - 15 40 - - 8f 1f- cc 1b cb 68 28 d0			

Use Azure Data Lake Storage

There are two ways the SDK can acquire the necessary OAuth token to access Azure Data Lake Storage:

- Retrieve the access token from a recent session of the user's Azure CLI login
 - Use a service principal (SP) and a certificate as secret

Use an access token from a recent Azure CLI session

On your local machine, run the following command.

```
az login  
az account show --query tenantId  
dataflow = read_csv(path = DataLakeDataSource(path='adl://dpreptestfiles.azuredatalakestore.net/farmers-  
markets.csv', tenant='microsoft.onmicrosoft.com')) head = dataflow.head(5) head
```

NOTE

If your user account is a member of more than one Azure tenant, you need to specify the tenant in the AAD URL hostname form.

Create a service principal with the Azure CLI

Use the Azure CLI to create a service principal and the corresponding certificate. This particular service principal is configured with the `reader` role, with its scope reduced to only the Azure Data Lake Storage account 'dpreptestfiles'.

```

az account set --subscription "Data Wrangling development"
az ad sp create-for-rbac -n "SP-ADLS-dpreptestfiles" --create-cert --role reader --scopes
/subscriptions/35f16a99-532a-4a47-9e93-
00305f6c40f2/resourceGroups/dpreptestfiles/providers/Microsoft.DataLakeStore/accounts/dpreptestfiles

```

This command emits the `appId` and the path to the certificate file (usually in the home folder). The .crt file contains both the public cert and the private key in PEM format.

```
openssl x509 -in adls-dpreptestfiles.crt -noout -fingerprint
```

To configure the ACL for the Azure Data Lake Storage file system, use the `objectId` of the user. In this example, the service principal is used.

```
az ad sp show --id "8dd38f34-1fcf-4ff9-accd-7cd60b757174" --query objectId
```

To configure `Read` and `Execute` access for the Azure Data Lake Storage file system, you configure the ACL for folders and files individually. This is due to the fact that the underlying HDFS ACL model doesn't support inheritance.

```

az dls fs access set-entry --account dpreptestfiles --acl-spec "user:e37b9b1f-6a5e-4bee-9def-402b956f4e6f:r-x"
--path /
az dls fs access set-entry --account dpreptestfiles --acl-spec "user:e37b9b1f-6a5e-4bee-9def-402b956f4e6f:r--"
--path /farmers-markets.csv

```

```

certThumbprint = 'C2:08:9D:9E:D1:74:FC:EB:E9:7E:63:96:37:1C:13:88:5E:B9:2C:84'
certificate = ''
with open('./data/adls-dpreptestfiles.crt', 'rt', encoding='utf-8') as crtFile:
    certificate = crtFile.read()

servicePrincipalAppId = "8dd38f34-1fcf-4ff9-accd-7cd60b757174"

```

Acquire an OAuth access token

Use the `adal` package (`pip install adal`) to create an authentication context on the MSFT tenant and acquire an OAuth access token. For ADLS, the resource in the token request must be for '<https://datalake.azure.net>', which is different from most other Azure resources.

```

import adal
from azureml.dataprep.api.datasources import DataLakeDataSource

ctx = adal.AuthenticationContext('https://login.microsoftonline.com/microsoft.onmicrosoft.com')
token = ctx.acquire_token_with_client_certificate('https://datalake.azure.net/', servicePrincipalAppId,
certificate, certThumbprint)
dataflow = dprep.read_csv(path = DataLakeDataSource(path='adl://dpreptestfiles.azuredatalakestore.net/farmers-
markets.csv', accessToken=token['accessToken']))
dataflow.to_pandas_dataframe().head()

```

	FMID	MARKETNAME	WEBSITE	STREET	CITY	COUNTY
0	1012063	Caledonia Farmers Market Association - Danville	https://sites.google.com/site/caledoniafarmers...		Danville	Caledonia

	FMID	MARKETNAME	WEBSITE	STREET	CITY	COUNTY
1	1011871	Stearns Homestead Farmers' Market	http://Stearns homestead.co m	6975 Ridge Road	Parma	Cuyahoga
2	1011878	100 Mile Market	http://www.pfc markets.com	507 Harrison St	Kalamazoo	Kalamazoo
3	1009364	106 S. Main Street Farmers Market	http://thetow nofsixmile.wor dpress.com/	106 S. Main Street	Six Mile	
4	1010691	10th Street Community Farmers Market	http://agrimissouri.com/mo-grown/grodet ail.php...	10th Street and Poplar	Lamar	Barton

Transform data with the Azure Machine Learning Data Prep SDK

12/10/2018 • 17 minutes to read • [Edit Online](#)

In this article, you learn different methods of loading data using the [Azure Machine Learning Data Prep SDK](#). The SDK offers functions make it simple to add columns, filter out unwanted rows or columns, and impute missing values.

Currently there are functions for the following tasks:

- Add column using an expression
 - Impute missing values
 - Derive column by example
 - Filtering
 - Custom Python transforms

Add column using an expression

The Azure Machine Learning Data Prep SDK includes `substring` expressions you can use to calculate a value from existing columns, and then put that value in a new column. In this example, you load data and try to add columns to that input data.

```
import azureml.dataprep as dprep

# loading data
dataflow = dprep.read_csv(path=r'data\crime0-10.csv')
dataflow.head(3)
```


ID	Case Number	Date	Block	IUCR	Primary Type	Description	Location	Domestic	Community Area	Community Code	X Coordinate	Y Coordinate	Year	Update Date	Latitude	Longitude	Location
1	1013297765	HY / 05/2022	01/22:00:00	04X0	BATTERY	SIMPLE	STREET	false	true	49B	1867370	1946271	2015	07/12/2022	-87.0514	(42.7560)	

Use the `substring(start, length)` expression to extract the prefix from the Case Number column and put that string in a new column, `Case Category`. Passing the `substring_expression` variable to the `expression` parameter creates a new calculated column that executes the expression on each record.

```
substring_expression = dprep.col('Case Number').substring(0, 2)
case_category = dataflow.add_column(new_column_name='Case Category',
                                     prior_column='Case Number',
                                     expression=substring_expression)
case_category.head(3)
```

ID	Case Number	Category	Date	Block	IUCR	Primary Type	Description	Arrest	...	Community Area	FBI Code	X Coordinate	Y Coordinate	Year	Updated On	Latitude	Longitude	Location	
0	101420094900	HYHY	07/05/2019	000XN/5020	0800T	THEFT	\$500 AND UNDER	STREET	false	41	10	06	1129230	1193315	20715	07/12/2022	12:42PM	1.90466	41.46666
1	1013297765	HYHY	07/05/2019	011XW/200	0460	BATTERY	SIMPLE	STREET	false	49	108B	1167370	1194671	20715	07/12/2022	12:45PM	-8.70595	42.00118	-8.70595

	CASE NUMBER	CLASS	DATA TYPE	BLOCK	ITEM	DESCRIPTION	LOCATION	COMMUNITY AREA	CODE	X COORDINATE	Y COORDINATE	UPDATER	LATITUDE	LONGITUDE	LOCATION
ID	20134209227503	HYY	DATE	TIME	TYPE	DESCRIPTION	ARREST	WARD	...	53	08B	YEAR	EDITION	TITUDE	LOCATION
2	0	H	0	1	0	B	D	S	f	t	...	9	5	0	0
0	Y	Y	7	2	4	A	O	T	a	r		3	8	0	7
1	/		1	8	T	M	R	ls	u	e			B	1	/
4	0	X	6	T	E	E	E	e						5	1
2	5	X	5	E	S	S	E								2
2	/	S	R	T	T	T	T								/
7	2	F	Y	I	C	C	C								2
0	3	0	R	O	B	B	B								0
		1	1	N	A	A	A								1
		5	1	T	T	T	T								5
		1	1	A	T	T	T								1
		:	2	V	E	R	R								2
		0	0	E	Y	Y	Y								4
		0	0	0	S	S	S								2
		0	0	0	I	I	I								4
		P	0	0	M	M	M								6
		M	0	0	P	P	P								P
					0	L	L								M
						E	E								

Use the `substring(start)` expression to extract only the number from the Case Number column and create a new column. Convert it to a numeric data type using the `to_number()` function, and pass the string column name as a parameter.

```
substring_expression2 = dprep.col('Case Number').substring(2)
case_id = dataflow.add_column(new_column_name='Case Id',
                               prior_column='Case Number',
                               expression=substring_expression2)

case_id = case_id.to_number('Case Id')
case_id.head(3)
```

ID	Case Number		Case ID		Date	Block	IUCR	Primary Type	Description	Location	Address	...	Community Area	Code	FBICode	X Coordinate	Y Coordinate	Year	Update	Latitude	Longitude	Location
	CASE NUMBER	CASE ID	CASE ID	CASE ID	CASE ID	CASE ID	CASE ID	Primary Type	Description	Location	Address	...	Community Area	Code	FBICode	X Coordinate	Y Coordinate	Year	Update	Latitude	Longitude	Location
0	101490490790	H29907770	39050770	00000000	00000000	00000000	00000000	THEFT	\$500000	STREET	false	...	41	10	06	11929230	1192923150	2015	0722015	1222015	41223060	41223060PM
1	1013929276552655	H29907770	39050770	00000000	00000000	00000000	00000000	BATTERY	SIMPLE	STREET	false	true	...	49	18B	1167370	11946271	2015	0722015	1222015	41223060	41223060PM

Impute missing values

The SDK can impute missing values in specified columns. In this example, you load latitude and longitude values and then try to impute missing values in the input data.

```
import azureml.dataprep as dprep

# loading input data
df = dprep.read_csv(r'data\crime0-10.csv')
df = df.keep_columns(['ID', 'Arrest', 'Latitude', 'Longitude'])
df = df.to_number(['Latitude', 'Longitude'])
df.head(5)
```

	ID	ARREST	LATITUDE	LONGITUDE
0	10140490	false	41.973309	-87.800175
1	10139776	false	42.008124	-87.659550
2	10140270	false	NaN	NaN

	ID	ARREST	LATITUDE	LONGITUDE
3	10139885	false	41.902152	-87.754883
4	10140379	false	41.885610	-87.657009

The third record is missing latitude and longitude values. To impute those missing values, you use

`ImputeMissingValuesBuilder` to learn a fixed expression. It can impute the columns with either a calculated `MIN`, `MAX`, `MEAN` value, or a `CUSTOM` value. When `group_by_columns` is specified, missing values will be imputed by group with `MIN`, `MAX`, and `MEAN` calculated per group.

Check the `MEAN` value of the latitude column using the `summarize()` function. This function accepts an array of columns in the `group_by_columns` parameter to specify the aggregation level. The `summary_columns` parameter accepts a `SummaryColumnsValue` call. This function call specifies the current column name, the new calculated field name, and the `SummaryFunction` to perform.

```
df_mean = df.summarize(group_by_columns=['Arrest'],
                       summary_columns=[dprep.SummaryColumnsValue(column_id='Latitude',
                                                               summary_column_name='Latitude_MEAN',
                                                               summary_function=dprep.SummaryFunction.MEAN)])
df_mean = df_mean.filter(dprep.col('Arrest') == 'false')
df_mean.head(1)
```

	ARREST	LATITUDE_MEAN
0	false	41.878961

The `MEAN` value of latitudes looks accurate, use the `ImputeColumnArguments` function to impute it. This function accepts a `column_id` string, and a `ReplaceValueFunction` to specify the impute type. For the missing longitude value, impute it with 42 based on external knowledge.

Impute steps can be chained together into a `ImputeMissingValuesBuilder` object, using the builder function `impute_missing_values()`. The `impute_columns` parameter accepts an array of `ImputeColumnArguments` objects. Call the `learn()` function to store the impute steps, and then apply to a dataflow object using `to_dataflow()`.

```
# impute with MEAN
impute_mean = dprep.ImputeColumnArguments(column_id='Latitude',
                                            impute_function=dprep.ReplaceValueFunction.MEAN)

# impute with custom value 42
impute_custom = dprep.ImputeColumnArguments(column_id='Longitude',
                                              custom_impute_value=42)

# get instance of ImputeMissingValuesBuilder
impute_builder = df.builders.impute_missing_values(impute_columns=[impute_mean, impute_custom],
                                                    group_by_columns=['Arrest'])

# call learn() to learn a fixed program to impute missing values
impute_builder.learn()

# call to_dataflow() to get a data flow with impute step added
df_imputed = impute_builder.to_dataflow()

# check impute result
df_imputed.head(5)
```

	ID	ARREST	LATITUDE	LONGITUDE
0	10140490	false	41.973309	-87.800175

	ID	ARREST	LATITUDE	LONGITUDE
1	10139776	false	42.008124	-87.659550
2	10140270	false	41.878961	42.000000
3	10139885	false	41.902152	-87.754883
4	10140379	false	41.885610	-87.657009

As shown in the result above, the missing latitude was imputed with the `MEAN` value of `Arrest=='false'` group. The missing longitude was imputed with 42.

```
imputed_longitude = df_imputed.to_pandas_dataframe()['Longitude'][2]
assert imputed_longitude == 42
```

Derive column by example

One of the more advanced tools in the Azure Machine Learning Data Prep SDK is the ability to derive columns using examples of desired results. This lets you give the SDK an example so it can generate code to achieve the intended transformation.

```
import azureml.dataprep as dprep
dataflow = dprep.read_csv(path='https://dpreptestfiles.blob.core.windows.net/testfiles/BostonWeather.csv')
dataflow.head(10)
```

	DATE	REPORTTPYE	HOURLYDRYBULB TEMPF	HOURLYRELATIVE HUMIDITY	HOURLYWINDSPEE D
0	1/1/2015 0:54	FM-15	22	50	10
1	1/1/2015 1:00	FM-12	22	50	10
2	1/1/2015 1:54	FM-15	22	50	10
3	1/1/2015 2:54	FM-15	22	50	11
4	1/1/2015 3:54	FM-15	24	46	13
5	1/1/2015 4:00	FM-12	24	46	13
6	1/1/2015 4:54	FM-15	22	52	15
7	1/1/2015 5:54	FM-15	23	48	17
8	1/1/2015 6:54	FM-15	23	50	14
9	1/1/2015 7:00	FM-12	23	50	14

Assume that you need to join this file with a dataset where date and time are in a format 'Mar 10, 2018 | 2AM-4AM'.

```
builder = dataflow.builders.derive_column_by_example(source_columns=['DATE'], new_column_name='date_timerange')
builder.add_example(source_data=df.iloc[1], example_value='Jan 1, 2015 12AM-2AM')
builder.preview()
```

	DATE	DATE_TIMERANGE
0	1/1/2015 0:54	Jan 1, 2015 12AM-2AM
1	1/1/2015 1:00	Jan 1, 2015 12AM-2AM
2	1/1/2015 1:54	Jan 1, 2015 12AM-2AM
3	1/1/2015 2:54	Jan 1, 2015 2AM-4AM
4	1/1/2015 3:54	Jan 1, 2015 2AM-4AM
5	1/1/2015 4:00	Jan 1, 2015 4AM-6AM
6	1/1/2015 4:54	Jan 1, 2015 4AM-6AM
7	1/1/2015 5:54	Jan 1, 2015 4AM-6AM
8	1/1/2015 6:54	Jan 1, 2015 6AM-8AM
9	1/1/2015 7:00	Jan 1, 2015 6AM-8AM

The code above first creates a builder for the derived column. You provide an array of source columns to consider (`DATE`), and a name for the new column to be added. As the first example, you pass in the second row (index 1) and give an expected value for the derived column.

Finally, you call `builder.preview()` and can see the derived column next to the source column. The format seems correct, but you only see values for the same date "Jan 1, 2015".

Now, pass in the number of rows you want to `skip` from the top to see rows further down.

```
builder.preview(skip=30)
```

	DATE	DATE_TIMERANGE
30	1/1/2015 22:54	Jan 1, 2015 10PM-12AM
31	1/1/2015 23:54	Jan 1, 2015 10PM-12AM
32	1/1/2015 23:59	Jan 1, 2015 10PM-12AM
33	1/2/2015 0:54	Feb 1, 2015 12AM-2AM
34	1/2/2015 1:00	Feb 1, 2015 12AM-2AM
35	1/2/2015 1:54	Feb 1, 2015 12AM-2AM
36	1/2/2015 2:54	Feb 1, 2015 2AM-4AM

	DATE	DATE_TIMERANGE
37	1/2/2015 3:54	Feb 1, 2015 2AM-4AM
38	1/2/2015 4:00	Feb 1, 2015 4AM-6AM
39	1/2/2015 4:54	Feb 1, 2015 4AM-6AM

Here you see an issue with the generated program. Based solely on the one example you provided above, the derive program chose to parse the date as "Day/Month/Year", which is not what you want in this case. To fix this issue, provide another example using the `add_example()` function on the `builder` variable.

```
builder.add_example(source_data=preview_df.iloc[3], example_value='Jan 2, 2015 12AM-2AM')
builder.preview(skip=30, count=10)
```

	DATE	DATE_TIMERANGE
30	1/1/2015 22:54	Jan 1, 2015 10PM-12AM
31	1/1/2015 23:54	Jan 1, 2015 10PM-12AM
32	1/1/2015 23:59	Jan 1, 2015 10PM-12AM
33	1/2/2015 0:54	Jan 2, 2015 12AM-2AM
34	1/2/2015 1:00	Jan 2, 2015 12AM-2AM
35	1/2/2015 1:54	Jan 2, 2015 12AM-2AM
36	1/2/2015 2:54	Jan 2, 2015 2AM-4AM
37	1/2/2015 3:54	Jan 2, 2015 2AM-4AM
38	1/2/2015 4:00	Jan 2, 2015 4AM-6AM
39	1/2/2015 4:54	Jan 2, 2015 4AM-6AM

Now rows correctly handle '1/2/2015' as 'Jan 2, 2015', but if you look further down the derived column, you see that values at the end have nothing in derived column. To fix that, you need to provide another example for row 66.

```
builder.add_example(source_data=preview_df.iloc[66], example_value='Jan 29, 2015 8PM-10PM')
builder.preview(count=10)
```

	DATE	DATE_TIMERANGE
0	1/1/2015 22:54	Jan 1, 2015 10PM-12AM
1	1/1/2015 23:54	Jan 1, 2015 10PM-12AM

	DATE	DATE_TIMERANGE
2	1/1/2015 23:59	Jan 1, 2015 10PM-12AM
3	1/2/2015 0:54	Jan 2, 2015 12AM-2AM
4	1/2/2015 1:00	Jan 2, 2015 12AM-2AM
5	1/2/2015 1:54	Jan 2, 2015 12AM-2AM
6	1/2/2015 2:54	Jan 2, 2015 2AM-4AM
7	1/2/2015 3:54	Jan 2, 2015 2AM-4AM
8	1/2/2015 4:00	Jan 2, 2015 4AM-6AM
9	1/2/2015 4:54	Jan 2, 2015 4AM-6AM

To separate date and time with '|', you add another example. This time, instead of passing in a row from the preview, construct a dictionary of column name to value for the `source_data` parameter.

```
builder.add_example(source_data={'DATE': '11/11/2015 0:54'}, example_value='Nov 11, 2015 | 12AM-2AM')
builder.preview(count=10)
```

	DATE	DATE_TIMERANGE
0	1/1/2015 22:54	None
1	1/1/2015 23:54	None
2	1/1/2015 23:59	None
3	1/2/2015 0:54	None
4	1/2/2015 1:00	None
5	1/2/2015 1:54	None
6	1/2/2015 2:54	None
7	1/2/2015 3:54	None
8	1/2/2015 4:00	None
9	1/2/2015 4:54	None

This clearly had negative effects, as now the only rows that have any values in derived column are the ones that match exactly with the examples we provided. Call `list_examples()` on the builder object to see a list of current example derivations.

```
examples = builder.list_examples()
```

	DATE	EXAMPLE	EXAMPLE_ID
0	1/1/2015 1:00	Jan 1, 2015 12AM-2AM	-1
1	1/2/2015 0:54	Jan 2, 2015 12AM-2AM	-2
2	1/29/2015 20:54	Jan 29, 2015 8PM-10PM	-3
3	11/11/2015 0:54	Nov 11, 2015 12AM-2AM	-4

In this case, inconsistent examples have been provided. To fix the issue, replace the first three examples with correct ones (including '|' between date and time).

Fix inconsistent examples by deleting examples that are incorrect (by either passing in `example_row` from the pandas DataFrame, or by passing in `example_id` value) and then adding new modified examples back.

```
builder.delete_example(example_id=-1)
builder.delete_example(example_row=examples.iloc[1])
builder.delete_example(example_row=examples.iloc[2])
builder.add_example(examples.iloc[0], 'Jan 1, 2015 | 12AM-2AM')
builder.add_example(examples.iloc[1], 'Jan 2, 2015 | 12AM-2AM')
builder.add_example(examples.iloc[2], 'Jan 29, 2015 | 8PM-10PM')
builder.preview()
```

	DATE	DATE_TIMERANGE
0	1/1/2015 0:54	Jan 1, 2015 12AM-2AM
1	1/1/2015 1:00	Jan 1, 2015 12AM-2AM
2	1/1/2015 1:54	Jan 1, 2015 12AM-2AM
3	1/1/2015 2:54	Jan 1, 2015 2AM-4AM
4	1/1/2015 3:54	Jan 1, 2015 2AM-4AM
5	1/1/2015 4:00	Jan 1, 2015 4AM-6AM
6	1/1/2015 4:54	Jan 1, 2015 4AM-6AM
7	1/1/2015 5:54	Jan 1, 2015 4AM-6AM
8	1/1/2015 6:54	Jan 1, 2015 6AM-8AM
9	1/1/2015 7:00	Jan 1, 2015 6AM-8AM

Now the data looks correct and you call `to_dataflow()` on the builder, which will return a data flow with the desired derived columns added.

```
dataflow = builder.to_dataflow()
df = dataflow.to_pandas_dataframe()
```

Filtering

The SDK includes the methods `Dataflow.drop_columns` and `Dataflow.filter` to let you filter out columns or rows.

Initial setup

```
import azureml.dataprep as dprep
from datetime import datetime
dataflow = dprep.read_csv(path='https://dprepdata.blob.core.windows.net/demo/green-small/*')
dataflow.head(5)
```

	LPEP_PIC_KUP_DATE_TIME	LPEP_DR_OPO_FF_D_ATETIME	STO_RE_AND_FWDFLAG	RATE_CODE_ID	PICK_UP_LONGITUDE	PICK_UP_LATITUDE	DROPOFF_LONGITUDE	DROPOFF_LATITUDE	PASS_ENGER_COUNT	TRIP_DISTANCE	TIP_AMOUNT	TOLL_SAMOUNT	TOTAL_AMOUNT
0	None	None	None	None	None	None	None	None	None	None	None	None	None
1	2013-08-01 08:14:37	2013-08-01 09:09:06	N	1	0	0	0	0	1	.00	0	0	21.25
2	2013-08-01 09:13:00	2013-08-01 11:38:00	N	1	0	0	0	0	2	.00	0	0	75
3	2013-08-01 09:48:00	2013-08-01 09:49:00	N	5	0	0	0	0	1	.00	0	1	2.1
4	2013-08-01 10:38:35	2013-08-01 10:38:51	N	1	0	0	0	0	1	.00	0	0	3.25

Filtering columns

To filter columns, use `Dataflow.drop_columns`. This method takes a list of columns to drop or a more complex argument called `ColumnSelector`.

Filtering columns with list of strings

In this example, `drop_columns` takes a list of strings. Each string should exactly match the desired column to drop.

```
dataflow = dataflow.drop_columns(['Store_and_fwd_flag', 'RateCodeID'])
dataflow.head(5)
```

	LPEP_PICKUP_DATETIME	LPEP_DROPOFF_DATETIME	PICKUP_LONITUDE	PICKUP_LATITUDE	DROP_OFF_LONGITUDE	DROP_OFF_LATITUDE	PASSENGER_COUNT	TRIP_DISTANCE	TIP_AMOUNT	TOLLS_AMOUNT	TOTAL_AMOUNT
0	None	None	None	None	None	None	None	None	None	None	None
1	2013-08-01 08:14:37	2013-08-01 09:09:06	0	0	0	0	1	.00	0	0	21.25
2	2013-08-01 09:13:00	2013-08-01 11:38:00	0	0	0	0	2	.00	0	0	75
3	2013-08-01 09:48:00	2013-08-01 09:49:00	0	0	0	0	1	.00	0	1	2.1
4	2013-08-01 10:38:35	2013-08-01 10:38:51	0	0	0	0	1	.00	0	0	3.25

Filtering columns with regex

Alternatively, use the `ColumnSelector` expression to drop columns that match a regex expression. In this example, you drop all the columns that match the expression `Column*|.*longitude|.*latitude`.

```
dataflow = dataflow.drop_columns(dprep.ColumnSelector('Column*|.*longitude|.*latitude', True, True))
dataflow.head(5)
```

	LPEP_PICKUP_DATETIME	LPEP_DROPOFF_DATETIME	PASSENGER_COUNT	TRIP_DISTANCE	TIP_AMOUNT	TOLLS_AMOUNT	TOTAL_AMOUNT
0	None	None	None	None	None	None	None
1	2013-08-01 08:14:37	2013-08-01 09:09:06	1	.00	0	0	21.25
2	2013-08-01 09:13:00	2013-08-01 11:38:00	2	.00	0	0	75
3	2013-08-01 09:48:00	2013-08-01 09:49:00	1	.00	0	1	2.1
4	2013-08-01 10:38:35	2013-08-01 10:38:51	1	.00	0	0	3.25

Filtering rows

To filter rows, use `DataFlow.filter`. This method takes an Azure Machine Learning Data Prep SDK expression as an argument, and returns a new data flow with the rows that the expression evaluates as True. Expressions are built using expression builders (`col`, `f_not`, `f_and`, `f_or`) and regular operators (`>`, `<`, `>=`, `<=`, `==`, `!=`).

Filtering rows with simple Expressions

Use the expression builder `col`, specify the column name as a string argument `col('column_name')`. Use this expression in combination with one of the following standard operators `>`, `<`, `>=`, `<=`, `==`, `!=` to build an expression such as `col('Tip_amount') > 0`. Finally, pass the built expression into the `Dataflow.filter` function.

In this example, `dataflow.filter(col('Tip_amount') > 0)` returns a new data flow with the rows in which the value of `Tip_amount` is greater than 0.

NOTE

`Tip_amount` is first converted to numeric, which allows us to build an expression comparing it against other numeric values.

```
dataflow = dataflow.to_number(['Tip_amount'])
dataflow = dataflow.filter(dprep.col('Tip_amount') > 0)
dataflow.head(5)
```

	LPEP_PICKUP_DATETIME	LPEP_DROPOFF_DATETIME	PASSENGER_COUNT	TRIP_DISTANCE	TIP_AMOUNT	TOLLS_AMOUNT	TOTAL_AMOUNT
0	2013-08-01 19:33:28	2013-08-01 19:35:21	5	.00	0.08	0	4.58
1	2013-08-05 13:16:38	2013-08-05 13:18:24	1	.00	0.30	0	3.8
2	2013-08-05 14:11:42	2013-08-05 14:12:47	1	.00	1.05	0	4.55
3	2013-08-05 14:15:56	2013-08-05 14:18:04	5	.00	2.22	0	5.72
4	2013-08-05 14:42:14	2013-08-05 14:42:38	1	.00	0.88	0	4.38

Filtering rows with complex expressions

To filter using complex expressions, combine one or more simple expressions with the expression builders `f_not`, `f_and`, or `f_or`.

In this example, `Dataflow.filter` returns a new data flow with the rows where `'Passenger_count'` is less than 5 and `'Tolls_amount'` is greater than 0.

```
dataflow = dataflow.to_number(['Passenger_count', 'Tolls_amount'])
dataflow = dataflow.filter(dprep.f_and(dprep.col('Passenger_count') < 5, dprep.col('Tolls_amount') > 0))
dataflow.head(5)
```

	LPEP_PICKUP_DATETIME	LPEP_DROPOFF_DATETIME	PASSENGER_COUNT	TRIP_DISTANCE	TIP_AMOUNT	TOLLS_AMOUNT	TOTAL_AMOUNT
0	2013-08-08 12:16:00	2013-08-08 12:16:00	1.0	.00	2.25	5.00	19.75
1	2013-08-12 14:43:53	2013-08-12 15:04:50	1.0	5.28	6.46	5.33	32.29

	LPEP_PICKUP_DATETIME	LPEP_DROPOFF_DATETIME	PASSENGER_COUNT	TRIP_DISTANCE	TIP_AMOUNT	TOLLS_AMOUNT	TOTAL_AMOUNT
2	2013-08-12 19:48:12	2013-08-12 20:03:42	1.0	5.50	1.00	10.66	30.66
3	2013-08-13 06:11:06	2013-08-13 06:30:28	1.0	9.57	7.47	5.33	44.8
4	2013-08-16 20:33:50	2013-08-16 20:48:50	1.0	5.63	3.00	5.33	27.83

It is also possible to filter rows combining more than one expression builder to create a nested expression.

NOTE

`lpep_pickup_datetime` and `Lpep_dropoff_datetime` are first converted to datetime, which allows us to build an expression comparing it against other datetime values.

```
dataflow = dataflow.to_datetime(['lpep_pickup_datetime', 'Lpep_dropoff_datetime'], ['%Y-%m-%d %H:%M:%S'])
dataflow = dataflow.to_number(['Total_amount', 'Trip_distance'])
mid_2013 = datetime(2013,7,1)
dataflow = dataflow.filter(
    dprep.f_and(
        dprep.f_or(
            dprep.col('lpep_pickup_datetime') > mid_2013,
            dprep.col('Lpep_dropoff_datetime') > mid_2013),
        dprep.f_and(
            dprep.col('Total_amount') > 40,
            dprep.col('Trip_distance') < 10)))
dataflow.head(5)
```

	LPEP_PICKUP_DATETIME	LPEP_DROPOFF_DATETIME	PASSENGER_COUNT	TRIP_DISTANCE	TIP_AMOUNT	TOLLS_AMOUNT	TOTAL_AMOUNT
0	2013-08-13 06:11:06+00:00	2013-08-13 06:30:28+00:00	1.0	9.57	7.47	5.33	44.80
1	2013-08-23 12:28:20+00:00	2013-08-23 12:50:28+00:00	2.0	8.22	8.08	5.33	40.41
2	2013-08-25 09:12:52+00:00	2013-08-25 09:34:34+00:00	1.0	8.80	8.33	5.33	41.66
3	2013-08-25 16:46:51+00:00	2013-08-25 17:13:55+00:00	2.0	9.66	7.37	5.33	44.20

	LPEP_PICKUP_DATETIME	LPEP_DROPOFF_DATETIME	PASSENGER_COUNT	TRIP_DISTANCE	TIP_AMOUNT	TOLLS_AMOUNT	TOTAL_AMOUNT
4	2013-08-25 17:42:11+0:00	2013-08-25 18:02:57+0:00	1.0	9.60	6.87	5.33	41.20

Custom Python transforms

There will always be scenarios when the easiest option for making a transformation is writing your own script. The SDK provides three extension points that you can use for custom Python scripts.

- New script column
- New script filter
- Transform partition

Each of the extensions is supported in both the scale-up and scale-out runtime. A key advantage of using these extension points is that you don't need to pull all of the data in order to create a data frame. Your custom Python code will run just like other transforms, at scale, by partition, and typically in parallel.

Initial data preparation

Start by loading some data from Azure Blob.

```
import azureml.dataprep as dprep
col = dprep.col

df =
dprep.read_csv(path='https://dpreptestfiles.blob.core.windows.net/testfiles/read_csv_duplicate_headers.csv',
skip_rows=1)
df.head(5)
```

	STNAM	FIPST	LEAID	LEANM10	NCESSCH	MAM_MTH00NUMVALID_1011
0	ALABAMA	1	101710	Hale County	10171002158	
1	ALABAMA	1	101710	Hale County	10171002162	
2	ALABAMA	1	101710	Hale County	10171002156	
3	ALABAMA	1	101710	Hale County	10171000588	2
4	ALABAMA	1	101710	Hale County	10171000589	

Trim down the data set and do some basic transforms.

```
df = df.keep_columns(['stnam', 'leanm10', 'ncessch', 'MAM_MTH00numvalid_1011'])
df = df.replace_na(columns=['leanm10', 'MAM_MTH00numvalid_1011'], custom_na_list='.')
df = df.to_number(['ncessch', 'MAM_MTH00numvalid_1011'])
df.head(5)
```

	STNAM	LEANM10	NCESSCH	MAM_MTH00NUMVALI D_1011
0	ALABAMA	Hale County	1.017100e+10	None
1	ALABAMA	Hale County	1.017100e+10	None
2	ALABAMA	Hale County	1.017100e+10	None
3	ALABAMA	Hale County	1.017100e+10	2
4	ALABAMA	Hale County	1.017100e+10	None

Look for null values using the following filter.

```
df.filter(col('MAM_MTH00numvalid_1011').is_null()).head(5)
```

	STNAM	LEANM10	NCESSCH	MAM_MTH00NUMVALI D_1011
0	ALABAMA	Hale County	1.017100e+10	None
1	ALABAMA	Hale County	1.017100e+10	None
2	ALABAMA	Hale County	1.017100e+10	None
3	ALABAMA	Hale County	1.017100e+10	None
4	ALABAMA	Hale County	1.017100e+10	None

Transform partition

Use a pandas function to replace all null values with a 0. This code will be run by partition, not on the entire data set at one time. This means that on a large data set, this code may run in parallel as the runtime processes the data, partition by partition.

The Python script must define a function called `transform()` that takes two arguments, `df` and `index`. The `df` argument will be a pandas dataframe that contains the data for the partition and the `index` argument is a unique identifier of the partition. The transform function can fully edit the passed in dataframe, but must return a dataframe. Any libraries that the Python script imports must exist in the environment where the dataflow is run.

```
df = df.transform_partition("""
def transform(df, index):
    df['MAM_MTH00numvalid_1011'].fillna(0,inplace=True)
    return df
""")
df.head(5)
```

	STNAM	LEANM10	NCESSCH	MAM_MTH00NUMVALI D_1011
0	ALABAMA	Hale County	1.017100e+10	0.0
1	ALABAMA	Hale County	1.017100e+10	0.0

	STNAM	LEANM10	NCESSCH	MAM_MTH00NUMVALID_1011
2	ALABAMA	Hale County	1.017100e+10	0.0
3	ALABAMA	Hale County	1.017100e+10	2.0
4	ALABAMA	Hale County	1.017100e+10	0.0

New script column

You can use Python code to create a new column that has the county name and the state name, and also to capitalize the state name. To do this, use the `new_script_column()` method on the data flow.

The Python script must define a function called `newvalue()` that takes a single argument `row`. The `row` argument is a dict (`key`:column name, `val`: current value) and will be passed to this function for each row in the data set. This function must return a value to be used in the new column. Any libraries that the Python script imports must exist in the environment where the dataflow is run.

```
df = df.new_script_column(new_column_name='county_state', insert_after='leanm10', script="""
def newvalue(row):
    return row['leanm10'] + ', ' + row['stnam'].title()
""")
df.head(5)
```

	STNAM	LEANM10	COUNTY_STATE	NCESSCH	MAM_MTH00NUMVALID_1011
0	ALABAMA	Hale County	Hale County, Alabama	1.017100e+10	0.0
1	ALABAMA	Hale County	Hale County, Alabama	1.017100e+10	0.0
2	ALABAMA	Hale County	Hale County, Alabama	1.017100e+10	0.0
3	ALABAMA	Hale County	Hale County, Alabama	1.017100e+10	2.0
4	ALABAMA	Hale County	Hale County, Alabama	1.017100e+10	0.0

New Script Filter

Build a Python expression to filter the data set to only rows where 'Hale' is not in the new `county_state` column.

The expression returns `True` if we want to keep the row, and `False` to drop the row.

```
df = df.new_script_filter("""
def includerow(row):
    val = row['county_state']
    return 'Hale' not in val
""")
df.head(5)
```

	STNAM	LEANM10	COUNTY_STATE	NCESSCH	MAM_MTH00NUM VALID_1011
0	ALABAMA	Jefferson County	Jefferson County, Alabama	1.019200e+10	1.0
1	ALABAMA	Jefferson County	Jefferson County, Alabama	1.019200e+10	0.0
2	ALABAMA	Jefferson County	Jefferson County, Alabama	1.019200e+10	0.0
3	ALABAMA	Jefferson County	Jefferson County, Alabama	1.019200e+10	0.0
4	ALABAMA	Jefferson County	Jefferson County, Alabama	1.019200e+10	0.0

Write data using the Azure Machine Learning Data Prep SDK

12/13/2018 • 4 minutes to read • [Edit Online](#)

In this article, you learn different methods to write data using the Azure Machine Learning Data Prep SDK. Output data can be written at any point in a dataflow, and writes are added as steps to the resulting data flow and are run every time the data flow is. Data is written to multiple partition files to allow parallel writes.

Since there are no limitations to how many write steps there are in a pipeline, you can easily add additional write steps to get intermediate results for troubleshooting or for other pipelines.

Each time you run a write step, a full pull of the data in the data flow occurs. For example, a data flow with three write steps will read and process every record in the data set three times.

Supported data types and location

The following file formats are supported

- Delimited files (CSV, TSV, etc.)
- Parquet files

Using the [Azure Machine Learning Data Prep python SDK](#), you can write data to:

- a local file system
- Azure Blob Storage
- Azure Data Lake Storage

Spark considerations

When running a data flow in Spark, you must write to an empty folder. Attempting to run a write to an existing folder results in a failure. Make sure your target folder is empty or use a different target location for each run, or the write will fail.

Monitoring write operations

For your convenience, a sentinel file named SUCCESS is generated once a write is completed. Its presence helps you identify when an intermediate write has completed without having to wait for the whole pipeline to complete.

Example write code

For this example, start by loading data into a data flow. You reuse this data with different formats.

```
import azureml.dataprep as dprep
t = dprep.auto_read_file('./data/fixed_width_file.txt')
t = t.to_number('Column3')
t.head(10)
```

Example output:

	COLUMN 1	COLUMN 2	COLUMN 3	COLUMN 4	COLUMN 5	COLUMN 6	COLUMN 7	COLUMN 8	COLUMN 9
0	10000.0	99999.0	None	NO	NO	ENRS	NaN	NaN	NaN
1	10003.0	99999.0	None	NO	NO	ENSO	NaN	NaN	NaN
2	10010.0	99999.0	None	NO	JN	ENJA	70933.0	-8667.0	90.0
3	10013.0	99999.0	None	NO	NO		NaN	NaN	NaN
4	10014.0	99999.0	None	NO	NO	ENSO	59783.0	5350.0	500.0
5	10015.0	99999.0	None	NO	NO	ENBL	61383.0	5867.0	3270.0
6	10016.0	99999.0	None	NO	NO		64850.0	11233.0	140.0
7	10017.0	99999.0	None	NO	NO	ENFR	59933.0	2417.0	480.0
8	10020.0	99999.0	None	NO	SV		80050.0	16250.0	80.0
9	10030.0	99999.0	None	NO	SV		77000.0	15500.0	120.0

Delimited file example

The following code uses the `write_to_csv` function to write data to a delimited file.

```
# Create a new data flow using `write_to_csv`
write_t = t.write_to_csv(directory_path=dprep.LocalFileOutput('./test_out/'))

# Run the data flow to begin the write operation.
write_t.run_local()

written_files = dprep.read_csv('./test_out/part-*')
written_files.head(10)
```

Example output:

	COLUMN 1	COLUMN 2	COLUMN 3	COLUMN 4	COLUMN 5	COLUMN 6	COLUMN 7	COLUMN 8	COLUMN 9
0	10000.0	99999.0	ERROR	NO	NO	ENRS	ERROR	ERROR	ERROR
1	10003.0	99999.0	ERROR	NO	NO	ENSO	ERROR	ERROR	ERROR
2	10010.0	99999.0	ERROR	NO	JN	ENJA	70933.0	-8667.0	90.0
3	10013.0	99999.0	ERROR	NO	NO		ERROR	ERROR	ERROR
4	10014.0	99999.0	ERROR	NO	NO	ENSO	59783.0	5350.0	500.0
5	10015.0	99999.0	ERROR	NO	NO	ENBL	61383.0	5867.0	3270.0
6	10016.0	99999.0	ERROR	NO	NO		64850.0	11233.0	140.0

	COLUMN 1	COLUMN 2	COLUMN 3	COLUMN 4	COLUMN 5	COLUMN 6	COLUMN 7	COLUMN 8	COLUMN 9
7	10017.0	99999.0	ERROR	NO	NO	ENFR	59933.0	2417.0	480.0
8	10020.0	99999.0	ERROR	NO	SV		80050.0	16250.0	80.0
9	10030.0	99999.0	ERROR	NO	SV		77000.0	15500.0	120.0

In the preceding output, several errors appear in the numeric columns because of numbers that were not parsed correctly. When written to CSV, null values are replaced with the string "ERROR" by default.

Add parameters as part of your write call and specify a string to use to represent null values.

```
write_t = t.write_to_csv(directory_path=dprep.LocalFileOutput('./test_out/'),
                        error='BadData',
                        na='NA')
write_t.run_local()
written_files = dprep.read_csv('./test_out/part-*')
written_files.head(10)
```

The preceding code produces this output:

	COLUMN 1	COLUMN 2	COLUMN 3	COLUMN 4	COLUMN 5	COLUMN 6	COLUMN 7	COLUMN 8	COLUMN 9
0	10000.0	99999.0	BadData	NO	NO	ENRS	BadData	BadData	BadData
1	10003.0	99999.0	BadData	NO	NO	ENSO	BadData	BadData	BadData
2	10010.0	99999.0	BadData	NO	JN	ENJA	70933.0	-8667.0	90.0
3	10013.0	99999.0	BadData	NO	NO		BadData	BadData	BadData
4	10014.0	99999.0	BadData	NO	NO	ENSO	59783.0	5350.0	500.0
5	10015.0	99999.0	BadData	NO	NO	ENBL	61383.0	5867.0	3270.0
6	10016.0	99999.0	BadData	NO	NO		64850.0	11233.0	140.0
7	10017.0	99999.0	BadData	NO	NO	ENFR	59933.0	2417.0	480.0
8	10020.0	99999.0	BadData	NO	SV		80050.0	16250.0	80.0
9	10030.0	99999.0	BadData	NO	SV		77000.0	15500.0	120.0

Parquet file example

Similar to `write_to_csv`, the `write_to_parquet` function returns a new data flow with a write Parquet step that is executed when the data flow runs.

```
write_parquet_t = t.write_to_parquet(directory_path=dprep.LocalFileOutput('./test_parquet_out/'),
                                     error='MiscreantData')
```

Run the data flow to start the write operation.

```

write_parquet_t.run_local()

written_parquet_files = dprep.read_parquet_file('./test_parquet_out/part-*')
written_parquet_files.head(10)

```

The preceding code produces this output:

	COLUMN 1	COLUMN 2	COLUMN 3	COLUMN 4	COLUMN 5	COLUMN 6	COLUMN 7	COLUMN 8	COLUMN 9
0	10000.0	99999.0	Miscrea ntData	NO	NO	ENRS	Miscrea ntData	Miscrea ntData	Miscrea ntData
1	10003.0	99999.0	Miscrea ntData	NO	NO	ENSO	Miscrea ntData	Miscrea ntData	Miscrea ntData
2	10010.0	99999.0	Miscrea ntData	NO	JN	ENJA	70933.0	-8667.0	90.0
3	10013.0	99999.0	Miscrea ntData	NO	NO		Miscrea ntData	Miscrea ntData	Miscrea ntData
4	10014.0	99999.0	Miscrea ntData	NO	NO	ENSO	59783.0	5350.0	500.0
5	10015.0	99999.0	Miscrea ntData	NO	NO	ENBL	61383.0	5867.0	3270.0
6	10016.0	99999.0	Miscrea ntData	NO	NO		64850.0	11233.0	140.0
7	10017.0	99999.0	Miscrea ntData	NO	NO	ENFR	59933.0	2417.0	480.0
8	10020.0	99999.0	Miscrea ntData	NO	SV		80050.0	16250.0	80.0
9	10030.0	99999.0	Miscrea ntData	NO	SV		77000.0	15500.0	120.0

Set up compute targets for model training

12/13/2018 • 17 minutes to read • [Edit Online](#)

With the Azure Machine Learning service, you can train your model on different compute resources. These compute resources, called **compute targets**, can be local or in the cloud. In this document, you will learn about the supported compute targets and how to use them.

A compute target is a resource where your training script is run, or your model is hosted when deployed as a web service. You can create and manage a compute target using the Azure Machine Learning SDK, Azure portal, or Azure CLI. If you have compute targets that were created through another service (for example, an HDInsight cluster), you can use them by attaching them to your Azure Machine Learning service workspace.

There are three broad categories of compute targets that Azure Machine Learning supports:

- **Local**: Your local machine, or a cloud-based VM that you use as a dev/experimentation environment.
- **Managed Compute**: Azure Machine Learning Compute is a compute offering that is managed by the Azure Machine Learning service. It allows you to easily create single- or multi-node compute for training, testing, and batch inferencing.
- **Attached Compute**: You can also bring your own Azure cloud compute and attach it to Azure Machine Learning. Read more below on supported compute types and how to use them.

Supported compute targets

Azure Machine Learning service has varying support across the various compute targets. A typical model development lifecycle starts with dev/experimentation on a small amount of data. At this stage, we recommend using a local environment. For example, your local computer or a cloud-based VM. As you scale up your training on larger data sets, or do distributed training, we recommend using Azure Machine Learning Compute to create a single- or multi-node cluster that autoscales each time you submit a run. You can also attach your own compute resource, although support for various scenarios may vary as detailed below:

COMPUTE TARGET	GPU ACCELERATION	AUTOMATED HYPERPARAMETER TUNING	AUTOMATED MACHINE LEARNING	Pipeline Friendly
Local computer	Maybe		✓	
Azure Machine Learning Compute	✓	✓	✓	✓
Remote VM	✓	✓	✓	✓
Azure Databricks			✓	✓*
Azure Data Lake Analytics				✓*
Azure HDInsight				✓

IMPORTANT

* Azure Databricks and Azure Data Lake Analytics can **only** be used in a pipeline. For more information on pipelines, see the [Pipelines in Azure Machine Learning](#) document.

IMPORTANT

Azure Machine Learning Compute must be created from within a workspace. You cannot attach existing instances to a workspace.

Other compute targets must be created outside Azure Machine Learning and then attached to your workspace.

NOTE

Some compute targets rely on Docker container images when training a model. The GPU base image must be used on Microsoft Azure Services only. For model training, these services are:

- Azure Machine Learning Compute
- Azure Kubernetes Service
- The Data Science Virtual Machine.

Workflow

The workflow for developing and deploying a model with Azure Machine Learning follows these steps:

1. Develop machine learning training scripts in Python.
2. Create and configure or attach an existing compute target.
3. Submit the training scripts to the compute target.
4. Inspect the results to find the best model.
5. Register the model in the model registry.
6. Deploy the model.

IMPORTANT

Your training script isn't tied to a specific compute target. You can train initially on your local computer, then switch compute targets without having to rewrite the training script.

TIP

Whenever you associate a compute target with your workspace, either by creating managed compute or attaching existing compute, you need to provide a name to your compute. This should be between 2 and 16 characters in length.

Switching from one compute target to another involves creating a [run configuration](#). The run configuration defines how to run the script on the compute target.

Training scripts

When you start a training run, a snapshot of the directory containing your training scripts is created and sent to the compute target. For more information, see [snapshots](#).

Local computer

When training locally, you use the SDK to submit the training operation. You can train using a user-managed or system-managed environment.

User-managed environment

In a user-managed environment, you are responsible for ensuring that all the necessary packages are available in the Python environment you choose to run the script in. The following code snippet is an example of configuring training for a user-managed environment:

```
from azureml.core.runconfig import RunConfiguration

# Editing a run configuration property on-fly.
run_config_user_managed = RunConfiguration()

run_config_user_managed.environment.python.user_managed_dependencies = True

# You can choose a specific Python environment by pointing to a Python path
#run_config.environment.python.interpreter_path = '/home/ninghai/miniconda3/envs/sdk2/bin/python'
```

System-managed environment

System-managed environments rely on conda to manage the dependencies. Conda creates a file named `conda_dependencies.yml` that contains a list of dependencies. You can then ask the system to build a new conda environment and run your scripts in it. System-managed environments can be reused later, as long as the `conda_dependencies.yml` files remains unchanged.

The initial setup up of a new environment can take several minutes to complete, depending on the size of the required dependencies. The following code snippet demonstrates creating a system-managed environment that depends on scikit-learn:

```
from azureml.core.runconfig import RunConfiguration
from azureml.core.conda_dependencies import CondaDependencies

run_config_system_managed = RunConfiguration()

run_config_system_managed.environment.python.user_managed_dependencies = False
run_config_system_managed.auto_prepare_environment = True

# Specify conda dependencies with scikit-learn

run_config_system_managed.environment.python.conda_dependencies = CondaDependencies.create(conda_packages=['scikit-learn'])
```

Azure Machine Learning Compute

Azure Machine Learning Compute is a managed compute infrastructure that allows the user to easily create single- to multi-node compute. It is created **within your workspace region** and is a resource that can be shared with other users in your workspace. It scales up automatically when a job is submitted, and can be put in an Azure Virtual Network. It executes in a **containerized environment**, packaging your model's dependencies in a Docker container.

You can use Azure Machine Learning Compute to distribute the training process across a cluster of CPU or GPU compute nodes in the cloud. For more information on the VM sizes that include GPUs, see the [GPU optimized virtual machine sizes](#) documentation.

NOTE

Azure Machine Learning Compute has default limits on things like the number of cores that can be allocated. For more information, see the [Manage and request quotas for Azure resources](#) document.

You can create Azure Machine Learning Compute on-demand when you schedule a run, or as a persistent resource.

Run-based creation

You can create Azure Machine Learning Compute as a compute target at run-time. In this case, the compute is automatically created for your run, scales up to max_nodes that you specify in your run config, and is then **deleted automatically** after the run completes.

IMPORTANT

Run-based creation of Azure Machine Learning compute is currently in Preview state. Do not use run-based creation if you are using Hyperparameter Tuning or Automated Machine Learning. If you need to use Hyperparameter Tuning or Automated Machine Learning, create the Azure Machine Learning compute before submitting a run.

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

#Let us first list the supported VM families for Azure Machine Learning Compute
AmlCompute.supported_vmsizes()

from azureml.core.runconfig import RunConfiguration

# create a new runconfig object
run_config = RunConfiguration()

# signal that you want to use AmlCompute to execute script.
run_config.target = "amlcompute"

# AmlCompute will be created in the same region as workspace. Set vm size for AmlCompute from the list
# returned above
run_config.amlcompute.vm_size = 'STANDARD_D2_V2'
```

Persistent compute (Basic)

A persistent Azure Machine Learning Compute can be reused across multiple jobs. It can be shared with other users in the workspace, and is kept between jobs.

To create a persistent Azure Machine Learning Compute resource, you specify the `vm_size` and `max_nodes` parameters. Azure Machine Learning then uses smart defaults for the rest of the parameters. For example, the compute is set to autoscale down to zero nodes when not used and to create dedicated VMs to run your jobs as needed.

- **vm_size:** VM family of the nodes created by Azure Machine Learning Compute.
- **max_nodes:** Maximum nodes to autoscale to while running a job on Azure Machine Learning Compute.

```

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

# Choose a name for your CPU cluster
cpu_cluster_name = "cpucluster"

# Verify that cluster does not exist already
try:
    cpu_cluster = ComputeTarget(workspace=ws, name=cpu_cluster_name)
    print('Found existing cluster, use it.')
except ComputeTargetException:
    compute_config = AmlCompute.provisioning_configuration(vm_size='STANDARD_D2_V2',
                                                           max_nodes=4)
    cpu_cluster = ComputeTarget.create(ws, cpu_cluster_name, compute_config)

cpu_cluster.wait_for_completion(show_output=True)

```

Persistent compute (Advanced)

You can also configure several advanced properties when creating Azure Machine Learning Compute. These properties allow you to create a persistent cluster of fixed size, or within an existing Azure Virtual Network in your subscription.

In addition to `vm_size` and `max_nodes`, you can use the following properties:

- **min_nodes**: Minimum nodes (default 0 nodes) to downscale to while running a job on Azure Machine Learning Compute.
- **vm_priority**: Choose between 'dedicated' (default) and 'lowpriority' VMs when creating Azure Machine Learning Compute. Low Priority VMs use Azure's excess capacity and are thus cheaper but risk your run being pre-empted.
- **idle_seconds_before_scaledown**: Idle time (default 120 seconds) to wait after run completion before autoscaling to min_nodes.
- **vnet_resourcegroup_name**: Resource group of the **existing** virtual network. Azure Machine Learning Compute is created within this virtual network.
- **vnet_name**: Name of virtual network. The virtual network must be in the same region as your Azure Machine Learning workspace.
- **subnet_name**: Name of subnet within the virtual network. Azure Machine Learning Compute resources will be assigned IP addresses from this subnet range.

TIP

When you create a persistent Azure Machine Learning Compute resource you also have the ability to update its properties such as the `min_nodes` or the `max_nodes`. Simply call the `update()` function for it.

```

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

# Choose a name for your CPU cluster
cpu_cluster_name = "cpucluster"

# Verify that cluster does not exist already
try:
    cpu_cluster = ComputeTarget(workspace=ws, name=cpu_cluster_name)
    print('Found existing cluster, use it.')
except ComputeTargetException:
    compute_config = AmlCompute.provisioning_configuration(vm_size='STANDARD_D2_V2',
                                                            vm_priority='lowpriority',
                                                            min_nodes=2,
                                                            max_nodes=4,
                                                            idle_seconds_before_scaledown='300',
                                                            vnet_resourcegroup_name='<my-resource-group>',
                                                            vnet_name='<my-vnet-name>',
                                                            subnet_name='<my-subnet-name>')
    cpu_cluster = ComputeTarget.create(ws, cpu_cluster_name, compute_config)

cpu_cluster.wait_for_completion(show_output=True)

```

Remote VM

Azure Machine Learning also supports bringing your own compute resource and attaching it to your workspace. One such resource type is an arbitrary remote VM as long as it is accessible from Azure Machine Learning service. It could be either an Azure VM or a remote server in your organization or on-premises. Specifically, given the IP address and credentials (username/password or SSH key), you can use any accessible VM for remote runs. You can use a system-built conda environment, an already existing Python environment, or a Docker container. Execution using Docker container requires that you have Docker Engine running on the VM. This functionality is especially useful when you want a more flexible, cloud-based dev/experimentation environment than your local machine.

TIP

We recommended using the Data Science Virtual Machine as the Azure VM of choice for this scenario. It is a pre-configured data science and AI development environment in Azure with a curated choice of tools and frameworks for full lifecycle of ML development. For more information on using the Data Science Virtual Machine with Azure Machine Learning, see the [Configure a development environment](#) document.

WARNING

Azure Machine Learning only supports virtual machines running Ubuntu. When creating a virtual machine or selecting an existing one, you must select one that uses Ubuntu.

The following steps use the SDK to configure a Data Science Virtual Machine (DSVM) as a training target:

- To attach an existing virtual machine as a compute target, you must provide the fully qualified domain name, login name, and password for the virtual machine. In the example, replace <fqdn> with public fully qualified domain name of the VM, or the public IP address. Replace <username> and <password> with the SSH user and password for the VM:

```

from azureml.core.compute import RemoteCompute, ComputeTarget

# Create compute config.
attach_config = RemoteCompute.attach_configuration(address = "ipaddress",
                                                    ssh_port=22,
                                                    username='<username>',
                                                    password="<password>")

# If using SSH instead of a password, use this:
#
#
#
#
#                                                    ssh_port=22,
#                                                    username='<username>',
#                                                    password=None,
#                                                    private_key_file="path-to-file",
#                                                    private_key_passphrase="passphrase")

# Attach the compute
compute = ComputeTarget.attach(ws, "attach-dsvm", attach_config)

compute.wait_for_completion(show_output=True)

```

2. Create a configuration for the DSVM compute target. Docker and conda are used to create and configure the training environment on DSVM:

```

from azureml.core.runconfig import RunConfiguration
from azureml.core.conda_dependencies import CondaDependencies

# Load the "cpu-dsvm.runconfig" file (created by the above attach operation) in memory
run_config = RunConfiguration(framework = "python")

# Set compute target to the Linux DSVM
run_config.target = compute_target_name

# Use Docker in the remote VM
run_config.environment.docker.enabled = True

# Use CPU base image
# If you want to use GPU in DSVM, you must also use GPU base Docker image
azureml.core.runconfig.DEFAULT_GPU_IMAGE
run_config.environment.docker.base_image = azureml.core.runconfig.DEFAULT_CPU_IMAGE
print('Base Docker image is:', run_config.environment.docker.base_image)

# Ask system to provision a new one based on the conda_dependencies.yml file
run_config.environment.python.user_managed_dependencies = False

# Prepare the Docker and conda environment automatically when used the first time.
run_config.prepare_environment = True

# specify CondaDependencies obj
run_config.environment.python.conda_dependencies = CondaDependencies.create(conda_packages=['scikit-learn'])

```

Azure Databricks

Azure Databricks is an Apache Spark-based environment in the Azure cloud. It can be used as a compute target when training models with an Azure Machine Learning pipeline.

IMPORTANT

An Azure Databricks compute target can only be used in a Machine Learning pipeline.

You must create an Azure Databricks workspace before using it to train your model. To create these resource, see the [Run a Spark job on Azure Databricks](#) document.

To attach Azure Databricks as a compute target, you must use the Azure Machine Learning SDK and provide the following information:

- **Compute name:** The name you want to assign to this compute resource.
- **Databricks workspace name:** The name of the Azure Databricks workspace.
- **Access token:** The access token used to authenticate to Azure Databricks. To generate an access token, see the [Authentication](#) document.

The following code demonstrates how to attach Azure Databricks as a compute target:

```
databricks_compute_name = os.environ.get("AML_DATABRICKS_COMPUTE_NAME", "<databricks_compute_name>")
databricks_workspace_name = os.environ.get("AML_DATABRICKS_WORKSPACE", "<databricks_workspace_name>")
databricks_resource_group = os.environ.get("AML_DATABRICKS_RESOURCE_GROUP", "<databricks_resource_group>")
databricks_access_token = os.environ.get("AML_DATABRICKS_ACCESS_TOKEN", "<databricks_access_token>")

try:
    databricks_compute = ComputeTarget(workspace=ws, name=databricks_compute_name)
    print('Compute target already exists')
except ComputeTargetException:
    print('compute not found')
    print('databricks_compute_name {}'.format(databricks_compute_name))
    print('databricks_workspace_name {}'.format(databricks_workspace_name))
    print('databricks_access_token {}'.format(databricks_access_token))

    # Create attach config
    attach_config = DatabricksCompute.attach_configuration(resource_group = databricks_resource_group,
                                                            workspace_name = databricks_workspace_name,
                                                            access_token = databricks_access_token)

    databricks_compute = ComputeTarget.attach(
        ws,
        databricks_compute_name,
        attach_config
    )

    databricks_compute.wait_for_completion(True)
```

Azure Data Lake Analytics

Azure Data Lake Analytics is a big data analytics platform in the Azure cloud. It can be used as a compute target when training models with an Azure Machine Learning pipeline.

IMPORTANT

An Azure Data Lake Analytics compute target can only be used in a Machine Learning pipeline.

You must create an Azure Data Lake Analytics account before using it to train your model. To create this resource, see the [Get started with Azure Data Lake Analytics](#) document.

To attach Data Lake Analytics as a compute target, you must use the Azure Machine Learning SDK and provide the following information:

- **Compute name:** The name you want to assign to this compute resource.
- **Resource Group:** The resource group that contains the Data Lake Analytics account.
- **Account name:** The Data Lake Analytics account name.

The following code demonstrates how to attach Data Lake Analytics as a compute target:

```

adla_compute_name = os.environ.get("AML_ADLA_COMPUTE_NAME", "<adla_compute_name>")
adla_resource_group = os.environ.get("AML_ADLA_RESOURCE_GROUP", "<adla_resource_group>")
adla_account_name = os.environ.get("AML_ADLA_ACCOUNT_NAME", "<adla_account_name>")

try:
    adla_compute = ComputeTarget(workspace=ws, name=adla_compute_name)
    print('Compute target already exists')
except ComputeTargetException:
    print('compute not found')
    print('adla_compute_name {}'.format(adla_compute_name))
    print('adla_resource_id {}'.format(adla_resource_group))
    print('adla_account_name {}'.format(adla_account_name))
    # create attach config
    attach_config = AdlaCompute.attach_configuration(resource_group = adla_resource_group,
                                                       account_name = adla_account_name)

# Attach ADLA
adla_compute = ComputeTarget.attach(
    ws,
    adla_compute_name,
    attach_config
)

adla_compute.wait_for_completion(True)

```

TIP

Azure Machine Learning pipelines can only work with data stored in the default data store of the Data Lake Analytics account. If the data you need to work with is in a non-default store, you can use a [DataTransferStep](#) to copy the data before training.

Azure HDInsight

HDInsight is a popular platform for big-data analytics. It provides Apache Spark, which can be used to train your model.

IMPORTANT

You must create the HDInsight cluster before using it to train your model. To create a Spark on HDInsight cluster, see the [Create a Spark Cluster in HDInsight](#) document.

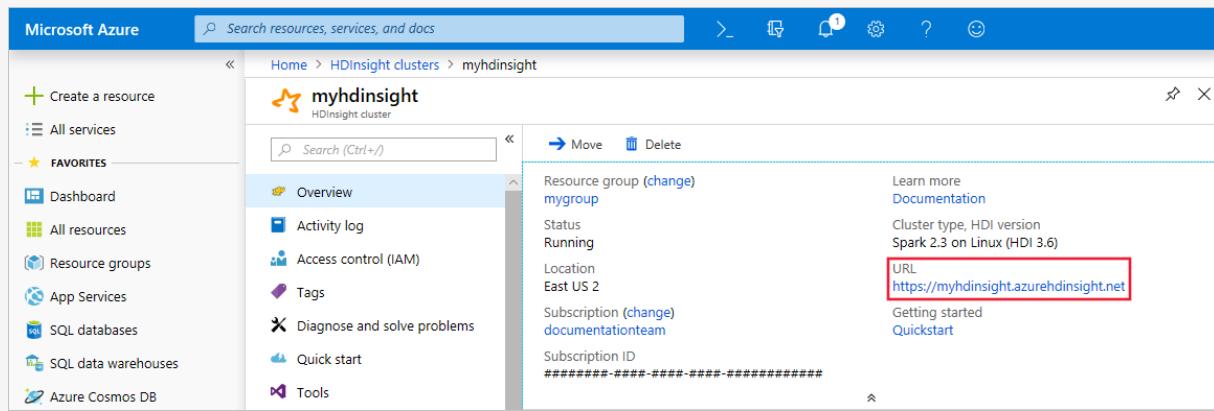
When creating the cluster, you must specify an SSH user name and password. Note these values, as you need them when using HDInsight as a compute target.

Once the cluster has been created, it has a fully qualified domain name (FQDN) of <clustername>.azurehdinsight.net , where <clustername> is the name you provided for the cluster. You need this address (or the public IP address of the cluster) to use it as a compute target

To configure HDInsight as a compute target, you must provide the fully qualified domain name, cluster login name, and password for the HDInsight cluster. The following example uses the SDK to attach a cluster to your workspace. In the example, replace <fqdn> with the public fully qualified domain name of the cluster, or the public IP address. Replace <username> and <password> with the SSH user and password for the cluster:

NOTE

To find the FQDN for your cluster, visit the Azure portal and select your HDInsight cluster. From the **Overview** section, the FQDN is part of the **URL** entry. Just remove the `https://` from the beginning of the value.



The screenshot shows the Azure portal interface with the search bar at the top. In the left sidebar, under 'FAVORITES', there are links for Dashboard, All resources, Resource groups, App Services, SQL databases, SQL data warehouses, and Azure Cosmos DB. The main content area shows a 'myhdinsight' HDInsight cluster. The 'Overview' tab is selected. On the right, there's a summary card with 'Resource group (change) mygroup', 'Status Running', 'Location East US 2', 'Subscription (change) documentationteam', 'Subscription ID ######-####-####-####-#####', and a 'URL' field containing `https://myhdinsight.azurehdinsight.net`. A red box highlights the URL field.

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

try:
    # Attaches a HDInsight cluster as a compute target.
    attach_config = HDInsightCompute.attach_configuration(address = "fqdn-or-ipaddress",
                                                          ssh_port = 22,
                                                          username = "username",
                                                          password = None, #if using ssh key
                                                          private_key_file = "path-to-key-file",
                                                          private_key_phrase = "key-phrase")

    compute = ComputeTarget.attach(ws, "myhdhi", attach_config)
except UserErrorException as e:
    print("Caught = {}".format(e.message))
    print("Compute config already attached.")

# Configure HDInsight run
# load the runconfig object from the "myhdhi.runconfig" file generated by the attach operaton above.
run_config = RunConfiguration.load(project_object = project, run_config_name = 'myhdhi')

# ask system to prepare the conda environment automatically when used for the first time
run_config.auto_prepare_environment = True
```

Submit training run

There are two ways to submit a training run:

- Submitting a `ScriptRunConfig` object.
- Submitting a `Pipeline` object.

IMPORTANT

The Azure Databricks and Azure Datalake Analytics compute targets can only be used in a pipeline. The local compute target cannot be used in a Pipeline.

Submit using `ScriptRunConfig`

The code pattern for submitting a training runs using `ScriptRunConfig` is the same regardless of the compute target:

- Create a `ScriptRunConfig` object using the run configuration for the compute target.

- Submit the run.
- Wait for the run to complete.

The following example uses the configuration for the system-managed local compute target created earlier in this document:

```
src = ScriptRunConfig(source_directory = script_folder, script = 'train.py', run_config =
run_config_system_managed)
run = exp.submit(src)
run.wait_for_completion(show_output = True)
```

Submit using a pipeline

The code pattern for submitting a training runs using a pipeline is the same regardless of the compute target:

- Add a step to the pipeline for the compute resource.
- Submit a run using the pipeline.
- Wait for the run to complete.

The following example uses the Azure Databricks compute target created earlier in this document:

```
dbStep = DatabricksStep(
    name="databricksmodule",
    inputs=[step_1_input],
    outputs=[step_1_output],
    num_workers=1,
    notebook_path=notebook_path,
    notebook_params={'myparam': 'testparam'},
    run_name='demo run name',
    databricks_compute=databricks_compute,
    allow_reuse=False
)
# list of steps to run
steps = [dbStep]
pipeline = Pipeline(workspace=ws, steps=steps)
pipeline_run = Experiment(ws, 'Demo_experiment').submit(pipeline)
pipeline_run.wait_for_completion()
```

For more information on machine learning pipelines, see the [Pipelines and Azure Machine Learning](#) document.

For example Jupyter Notebooks that demonstrate training using a pipeline, see <https://github.com/Azure/MachineLearningNotebooks/tree/master/pipeline>.

View and set up compute using the Azure portal

You can view what compute targets are associated with your workspace from the Azure portal. To get to the list, use the following steps:

1. Visit the [Azure portal](#) and navigate to your workspace.
2. Click on the **Compute** link under the **Applications** section.

The screenshot shows the Azure Machine Learning service workspace 'get-started-space'. On the left, there's a navigation sidebar with sections like Overview, Activity log, Access control (IAM), Tags, Diagnose and solve problems, Settings, Locks, Automation script, Properties, Application, Experiments, Pipelines, and Compute. The 'Compute' option is highlighted with a red box. Below the sidebar, there's a 'Getting Started' section with links to Explore your Azure Machine Learning service workspace, View Documentation, Get Started in Azure Notebooks, and View more samples at GitHub.

Create a compute target

Follow the above steps to view the list of compute targets, and then use the following steps to create a compute target:

1. Click the + sign to add a compute target.

The screenshot shows the 'Compute' page. It has a header with 'Compute', a 'Add Compute' button (highlighted with a red box), a 'Refresh' button, and a 'Delete' button. Below the header, it says 'There are no computers in this workspace.'

2. Enter a name for the compute target
3. Select **Machine Learning Compute** as the type of compute to use for **Training**

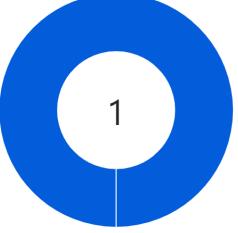
IMPORTANT

You can only create Azure Machine Learning Compute as the managed compute for training

4. Fill out the required form especially the VM Family and the maximum nodes to use for spinning up the compute
5. Select **Create**
6. You can view the status of the create operation by selecting the compute target from the list

Experiments	Pipelines	Compute	Models	Images	Deployments	Activities
Compute						
+ Add Compute	Refresh	Delete	Detach			
NAME ↑	TYPE ↑	PROVISIONING STATE ↑	CREATED DATE ↑	VIRTUAL MACHINE SIZE		
compute1	Machine Learning Compute	Succeeded	Mon Nov 19 2018	STANDARD_D1_V2		

7. You will then see the details for the compute target.

Experiments	Pipelines	Compute	Models	Images	Deployments	Activities
compute1						
Back to Compute List	Refresh	Edit	Delete	Detach		
CLUSTER NODE STATUS	CLUSTER STATE					
 <input type="checkbox"/> Show usable nodes only	Allocation state : steady Allocation state transition time : November 28, 2018, 2:18 AM Provisioning state : Succeeded Created : November 28, 2018, 2:18 AM Current node count : 1					
ATTRIBUTES	RESOURCE PROPERTIES					
Compute name: compute1	Virtual Machine size: STANDARD_D1_V2					
Compute type: Machine Learning Compute	Virtual Machine priority: dedicated					
Subscription ID	Minimum number of nodes: 1					
Resource group: pgunda	Maximum number of nodes: 1					
Workspace: pgeastus	Idle seconds before scale down: 120					
Region: eastus	Subnet: --					

8. Now you can submit a run against these targets as detailed above

Reuse existing compute in your workspace

Follow the above steps to view the list of compute targets, then use the following steps to reuse compute target:

1. Click the + sign to add a compute target
2. Enter a name for the compute target
3. Select the type of compute to attach for **Training**

IMPORTANT

Not all compute types can be attached using the portal. Currently the types that can be attached for training are:

- Remote VM
- Databricks
- Data Lake Analytics
- HDInsight

4. Fill out the required form

NOTE

Microsoft recommends that you use SSH keys, as they are more secure than passwords. Passwords are vulnerable to brute force attacks, while SSH keys rely on cryptographic signatures. For information on creating SSH keys for use with Azure Virtual Machines, see the following documents:

- [Create and use SSH keys on Linux or macOS](#)
- [Create and use SSH keys on Windows](#)

5. Select Attach
6. You can view the status of the attach operation by selecting the compute target from the list
7. Now you can submit a run against these targets as detailed above

Examples

Refer to notebooks in the following locations:

- [how-to-use-azureml/training](#)
- [tutorials/img-classification-part1-training.ipynb](#)

Learn how to run notebooks by following the article, [Use Jupyter notebooks to explore this service](#).

Next steps

- [Azure Machine Learning SDK reference](#)
- [Tutorial: Train a model](#)
- [Where to deploy models](#)
- [Build machine learning pipelines with Azure Machine Learning service](#)

Access data during training from your datastores

12/10/2018 • 3 minutes to read • [Edit Online](#)

Use a datastore to access and interact with your data in Azure Machine Learning workflows.

In Azure Machine Learning service, the datastore is an abstraction over [Azure Storage](#). The datastore can reference either an [Azure Blob](#) container or [Azure file share](#) as the underlying storage.

Create a datastore

To use datastores, you first need a [workspace](#). Start by either [creating a new workspace](#) or retrieving an existing one:

```
import azureml.core  
from azureml.core import Workspace  
  
ws = Workspace.from_config()
```

Use the default datastore

No need to create or configure a storage account. Each workspace has a default datastore that you can start using immediately.

To get the workspace's default datastore:

```
ds = ws.get_default_datastore()
```

Register a datastore

If you have existing Azure Storage, you can register it as a datastore on your workspace. You can register an Azure Blob Container or Azure File Share as a datastore. All the register methods are on the `Datastore` class and have the form `register_azure_*`.

Azure Blob Container Datastore

To register an Azure Blob Container datastore:

Azure File Share Datastore

To register an Azure File Share datastore:

Get an existing datastore

To query for a registered datastore by name:

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

You can also get all the datastores for a workspace:

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

For convenience, set one of your registered datastores as the default datastore for your workspace:

```
ws.set_default_datastore('your datastore name')
```

Upload and download data

Upload

Upload either a directory or individual files to the datastore using the Python SDK.

To upload a directory to a datastore `ds`:

```
ds.upload(src_dir='your source directory',
          target_path='your target path',
          overwrite=True,
          show_progress=True)
```

`target_path` specifies the location in the file share (or blob container) to upload. It defaults to `None`, in which case the data gets uploaded to root. `overwrite=True` will overwrite any existing data at `target_path`.

Or upload a list of individual files to the datastore via the datastore's `upload_files()` method.

Download

Similarly, download data from a datastore to your local file system.

```
ds.download(target_path='your target path',
            prefix='your prefix',
            show_progress=True)
```

`target_path` is the location of the local directory to download the data to. To specify a path to the folder in the file share (or blob container) to download, provide that path to `prefix`. If `prefix` is `None`, all the contents of your file share (or blob container) will get downloaded.

Access datastores for training

You can access a datastore during a training run (for example, for training or validation data) on a remote compute target via the Python SDK.

There are two supported ways to make your datastore available on the remote compute:

- **Mount**

`ds.as_mount()`: by specifying this mount mode, the datastore will get mounted for you on the remote compute.

- **Download/upload**

- `ds.as_download(path_on_compute='your path on compute')` downloads data from your datastore to the remote compute to the location specified by `path_on_compute`.
- `ds.as_upload(path_on_compute='yourfilename')` uploads data to the datastore. Suppose your training script creates a `foo.pkl` file in the current working directory on the remote compute. Upload this file to your datastore with `ds.as_upload(path_on_compute='./foo.pkl')` after the script creates the file. The file is uploaded to the root of your datastore.

To reference a specific folder or file in your datastore, use the datastore's `path` function. For example, to download the contents of the `./bar` directory from the datastore to your compute target, use

```
ds.path('./bar').as_download()
```

Any `ds` or `ds.path` object resolves to an environment variable name of the format `"$AZUREML_DATAREFERENCE_XXXX"` whose value represents the mount/download path on the remote compute. The datastore path on the remote compute might not be the same as the execution path for the script.

To access your datastore during training, pass it into your training script as a command-line argument via

```
script_params:
```

```
from azureml.train.estimator import Estimator

script_params = {
    '--data_dir': ds.as_mount()
}

est = Estimator(source_directory='your code directory',
                script_params=script_params,
                compute_target=compute_target,
                entry_script='train.py')
```

`as_mount()` is the default mode for a datastore, so you could also directly just pass `ds` to the `--data_dir` argument.

Or pass in a list of datastores to the Estimator constructor `inputs` parameter to mount or copy to/from your datastore(s):

```
est = Estimator(source_directory='your code directory',
                compute_target=compute_target,
                entry_script='train.py',
                inputs=[ds1.as_download(), ds2.path('./foo').as_download(),
                        ds3.as_upload(path_on_compute='./bar.pkl')])
```

The above code will:

- download all the contents in datastore `ds1` to the remote compute before your training script `train.py` is run
- download the folder `'./foo'` in datastore `ds2` to the remote compute before `train.py` is run
- upload the file `'./bar.pkl'` from the remote compute up to the datastore `ds3` after your script has run

Next steps

- [Train a model](#)

Train models with Azure Machine Learning

12/10/2018 • 4 minutes to read • [Edit Online](#)

Training machine learning models, particularly deep neural networks, is often a time- and compute-intensive task. Once you've finished writing your training script and running on a small subset of data on your local machine, you will likely want to scale up your workload.

To facilitate training, the Azure Machine Learning Python SDK provides a high-level abstraction, the estimator class, which allows users to easily train their models in the Azure ecosystem. You can create and use an [Estimator object](#) to submit any training code you want to run on remote compute, whether it's a single-node run or distributed training across a GPU cluster. For PyTorch and TensorFlow jobs, Azure Machine Learning also provides respective custom [PyTorch](#) and [TensorFlow](#) estimators to simplify using these frameworks.

Train with an estimator

Once you've created your [workspace](#) and set up your [development environment](#), training a model in Azure Machine Learning involves the following steps:

1. Create a [remote compute target](#)
2. Upload your [training data](#) (Optional)
3. Create your [training script](#)
4. Create an [Estimator](#) object
5. Submit your training job

This article focuses on steps 4-5. For steps 1-3, refer to the [train a model tutorial](#) for an example.

Single-node training

Use an [Estimator](#) for a single-node training run on remote compute in Azure for a scikit-learn model. You should have already created your [compute target](#) object `compute_target` and your [datastore](#) object `ds`.

```
from azureml.train.estimator import Estimator

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

sk_est = Estimator(source_directory='./my-sklearn-proj',
                    script_params=script_params,
                    compute_target=compute_target,
                    entry_script='train.py',
                    conda_packages=['scikit-learn'])
```

This code snippet specifies the following parameters to the [Estimator](#) constructor.

PARAMETER	DESCRIPTION
<code>source_directory</code>	Local directory that contains all of your code needed for the training job. This folder gets copied from your local machine to the remote compute

PARAMETER	DESCRIPTION
<code>script_params</code>	Dictionary specifying the command-line arguments to your training script <code>entry_script</code> , in the form of <command-line argument, value> pairs
<code>compute_target</code>	Remote compute target that your training script will run on, in this case an Azure Machine Learning Compute (AmlCompute) cluster
<code>entry_script</code>	Filepath (relative to the <code>source_directory</code>) of the training script to be run on the remote compute. This file, and any additional files it depends on, should be located in this folder
<code>conda_packages</code>	List of Python packages to be installed via conda needed by your training script.

The constructor has another parameter called `pip_packages` that you use for any pip packages needed

Now that you've created your `Estimator` object, submit the training job to be run on the remote compute with a call to the `submit` function on your `Experiment` object `experiment`.

```
run = experiment.submit(sk_est)
print(run.get_details().status)
```

IMPORTANT

Special Folders Two folders, *outputs* and *logs*, receive special treatment by Azure Machine Learning. During training, when you write files to folders named *outputs* and *logs* that are relative to the root directory (`./outputs` and `./logs`, respectively), the files will automatically upload to your run history so that you have access to them once your run is finished.

To create artifacts during training (such as model files, checkpoints, data files, or plotted images) write these to the `./outputs` folder.

Similarly, you write any logs from your training run to the `./logs` folder. To utilize Azure Machine Learning's [TensorBoard integration](#) make sure you write your TensorBoard logs to this folder. While your run is in progress, you will be able to launch TensorBoard and stream these logs. Later, you will also be able to restore the logs from any of your previous runs.

For example, to download a file written to the *outputs* folder to your local machine after your remote training run:

```
run.download_file(name='outputs/my_output_file', output_file_path='my_destination_path')
```

Distributed training and custom Docker images

There are two additional training scenarios you can carry out with the `Estimator`:

- Using a custom Docker image
- Distributed training on a multi-node cluster

The following code shows how to carry out distributed training for a CNTK model. In addition, instead of using the default Azure Machine Learning images, it assumes you are using your own custom docker image for training.

You should have already created your `compute target` object `compute_target`. You create the estimator as follows:

```

from azureml.train.estimator import Estimator

estimator = Estimator(source_directory='./my-cntk-proj',
                      compute_target=compute_target,
                      entry_script='train.py',
                      node_count=2,
                      process_count_per_node=1,
                      distributed_backend='mpi',
                      pip_packages=['cntk==2.5.1'],
                      custom_docker_base_image='microsoft/mmlspark:0.12')

```

The above code exposes the following new parameters to the `Estimator` constructor:

PARAMETER	DESCRIPTION	DEFAULT
<code>custom_docker_base_image</code>	Name of the image you want to use. Only provide images available in public docker repositories (in this case Docker Hub). To use an image from a private docker repository, use the constructor's <code>environment_definition</code> parameter instead	<code>None</code>
<code>node_count</code>	Number of nodes to use for your training job.	<code>1</code>
<code>process_count_per_node</code>	Number of processes (or "workers") to run on each node. In this case, you use the <code>2</code> GPUs available on each node.	<code>1</code>
<code>distributed_backend</code>	Backend for launching distributed training, which the Estimator offers via MPI. To carry out parallel or distributed training (e.g. <code>node_count > 1</code> or <code>process_count_per_node > 1</code> or both), set <code>distributed_backend='mpi'</code> . The MPI implementation used by AML is Open MPI .	<code>None</code>

Finally, submit the training job:

```
run = experiment.submit(cntk_est)
```

Examples

For a notebook that trains an sklearn model, see:

- [tutorials/img-classification-part1-training.ipynb](#)

For notebooks on distributed deep learning, see:

- [how-to-use-azureml/training-with-deep-learning](#)

Learn how to run notebooks by following the article, [Use Jupyter notebooks to explore this service](#).

Next steps

- Track run metrics during training
- Train PyTorch models
- Train TensorFlow models
- Tune hyperparameters
- Deploy a trained model

Train PyTorch models with Azure Machine Learning service

12/7/2018 • 2 minutes to read • [Edit Online](#)

For deep neural network (DNN) training using PyTorch, Azure Machine Learning provides a custom `PyTorch` class of the `Estimator`. The Azure SDK's `PyTorch` estimator enables you to easily submit PyTorch training jobs for both single-node and distributed runs on Azure compute.

Single-node training

Training with the `PyTorch` estimator is similar to using the base `Estimator`, so first read through the how-to article and make sure you understand the concepts introduced there.

To run a PyTorch job, instantiate a `PyTorch` object. You should have already created your `compute_target` object `compute_target` and your `datastore` object `ds`.

```
from azureml.train.dnn import PyTorch

script_params = {
    '--data_dir': ds
}

pt_est = PyTorch(source_directory='./my-pytorch-proj',
                  script_params=script_params,
                  compute_target=compute_target,
                  entry_script='train.py',
                  use_gpu=True)
```

Here, we specify the following parameters to the PyTorch constructor:

PARAMETER	DESCRIPTION
<code>source_directory</code>	Local directory that contains all of your code needed for the training job. This folder gets copied from your local machine to the remote compute
<code>script_params</code>	Dictionary specifying the command-line arguments to your training script <code>entry_script</code> , in the form of <command-line argument, value> pairs
<code>compute_target</code>	Remote compute target that your training script will run on, in this case an Azure Machine Learning Compute (AmlCompute) cluster
<code>entry_script</code>	Filepath (relative to the <code>source_directory</code>) of the training script to be run on the remote compute. This file, and any additional files it depends on, should be located in this folder
<code>conda_packages</code>	List of Python packages to be installed via conda needed by your training script. The constructor has another parameter called <code>pip_packages</code> that you can use for any pip packages needed

PARAMETER	DESCRIPTION
<code>use_gpu</code>	Set this flag to <code>True</code> to leverage the GPU for training. Defaults to <code>False</code>

Since you are using the `PyTorch` estimator, the container used for training will include the PyTorch package and related dependencies needed for training on CPUs and GPUs.

Then, submit the PyTorch job:

```
run = exp.submit(pt_est)
```

Distributed training

The `PyTorch` estimator also enables you to train your models at scale across CPU and GPU clusters of Azure VMs. You can easily run distributed PyTorch training with a few API calls, while Azure Machine Learning will manage behind the scenes all the infrastructure and orchestration needed to carry out these workloads.

Azure Machine Learning currently supports MPI-based distributed training of PyTorch using the Horovod framework.

Horovod

[Horovod](#) is an open-source ring-allreduce framework for distributed training developed by Uber.

To run distributed PyTorch using the Horovod framework, create the PyTorch object as follows:

```
from azureml.train.dnn import PyTorch

pt_est = PyTorch(source_directory='./my-pytorch-project',
                 script_params={},
                 compute_target=compute_target,
                 entry_script='train.py',
                 node_count=2,
                 process_count_per_node=1,
                 distributed_backend='mpi',
                 use_gpu=True)
```

This code exposes the following new parameters to the PyTorch constructor:

PARAMETER	DESCRIPTION	DEFAULT
<code>node_count</code>	Number of nodes to use for your training job.	<code>1</code>
<code>process_count_per_node</code>	Number of processes (or "workers") to run on each node.	<code>1</code>
<code>distributed_backend</code>	Backend for launching distributed training, which the Estimator offers via MPI. To carry out parallel or distributed training (e.g. <code>node_count</code> > 1 or <code>process_count_per_node</code> > 1 or both) with MPI (and Horovod), set <code>distributed_backend='mpi'</code> . The MPI implementation used by Azure Machine Learning is Open MPI .	<code>None</code>

The above example will run distributed training with two workers, one worker per node.

Horovod and its dependencies will be installed for you, so you can simply import it in your training script `train.py` as follows:

```
import torch
import horovod
```

Finally, submit your distributed PyTorch job:

```
run = exp.submit(pt_est)
```

Examples

For notebooks on distributed deep learning, see:

- [how-to-use-azureml/training-with-deep-learning](#)

Learn how to run notebooks by following the article, [Use Jupyter notebooks to explore this service](#).

Next steps

- [Track run metrics during training](#)
- [Tune hyperparameters](#)
- [Deploy a trained model](#)

Train TensorFlow models with Azure Machine Learning service

12/7/2018 • 4 minutes to read • [Edit Online](#)

For deep neural network (DNN) training using TensorFlow, Azure Machine Learning provides a custom `TensorFlow` class of the `Estimator`. The Azure SDK's `TensorFlow` estimator (not to be conflated with the `tf.estimator.Estimator` class) enables you to easily submit TensorFlow training jobs for both single-node and distributed runs on Azure compute.

Single-node training

Training with the `TensorFlow` estimator is similar to using the base `Estimator`, so first read through the how-to article and make sure you understand the concepts introduced there.

To run a TensorFlow job, instantiate a `TensorFlow` object. You should have already created your `compute target` object `compute_target`.

```
from azureml.train.dnn import TensorFlow

script_params = {
    '--batch-size': 50,
    '--learning-rate': 0.01,
}

tf_est = TensorFlow(source_directory='./my-tf-proj',
                    script_params=script_params,
                    compute_target=compute_target,
                    entry_script='train.py',
                    conda_packages=['scikit-learn'],
                    use_gpu=True)
```

Here, we specify the following parameters to the `TensorFlow` constructor:

PARAMETER	DESCRIPTION
<code>source_directory</code>	Local directory that contains all of your code needed for the training job. This folder gets copied from your local machine to the remote compute
<code>script_params</code>	Dictionary specifying the command-line arguments to your training script <code>entry_script</code> , in the form of <command-line argument, value> pairs
<code>compute_target</code>	Remote compute target that your training script will run on, in this case an Azure Machine Learning Compute (AmlCompute) cluster
<code>entry_script</code>	Filepath (relative to the <code>source_directory</code>) of the training script to be run on the remote compute. This file, and any additional files it depends on, should be located in this folder

PARAMETER	DESCRIPTION
conda_packages	List of Python packages to be installed via conda needed by your training script. In this case training script uses <code>sklearn</code> for loading the data, so specify this package to be installed. The constructor has another parameter called <code>pip_packages</code> that you can use for any pip packages needed
use_gpu	Set this flag to <code>True</code> to leverage the GPU for training. Defaults to <code>False</code> .

Since you are using the TensorFlow estimator, the container used for training will default include the TensorFlow package and related dependencies needed for training on CPUs and GPUs.

Then, submit the TensorFlow job:

```
run = exp.submit(tf_est)
```

Distributed training

The TensorFlow Estimator also enables you to train your models at scale across CPU and GPU clusters of Azure VMs. You can easily run distributed TensorFlow training with a few API calls, while Azure Machine Learning will manage behind the scenes all the infrastructure and orchestration needed to carry out these workloads.

Azure Machine Learning supports two methods of distributed training in TensorFlow:

- MPI-based distributed training using the [Horovod](#) framework
- Native [distributed TensorFlow](#) via the parameter server method

Horovod

[Horovod](#) is an open-source ring-allreduce framework for distributed training developed by Uber.

To run distributed TensorFlow using the Horovod framework, create the TensorFlow object as follows:

```
from azureml.train.dnn import TensorFlow

tf_est = TensorFlow(source_directory='./my-tf-proj',
                    script_params={},
                    compute_target=compute_target,
                    entry_script='train.py',
                    node_count=2,
                    process_count_per_node=1,
                    distributed_backend='mpi',
                    use_gpu=True)
```

The above code exposes the following new parameters to the TensorFlow constructor:

PARAMETER	DESCRIPTION	DEFAULT
node_count	Number of nodes to use for your training job.	1
process_count_per_node	Number of processes (or "workers") to run on each node.	1

PARAMETER	DESCRIPTION	DEFAULT
<code>distributed_backend</code>	Backend for launching distributed training, which the Estimator offers via MPI. If you want to carry out parallel or distributed training (e.g. <code>node_count</code> >1 or <code>process_count_per_node</code> >1 or both) with MPI (and Horovod), set <code>distributed_backend='mpi'</code> . The MPI implementation used by Azure Machine Learning is Open MPI .	None

The above example will run distributed training with two workers, one worker per node.

Horovod and its dependencies will be installed for you, so you can simply import it in your training script `train.py` as follows:

```
import tensorflow as tf
import horovod
```

Finally, submit the TensorFlow job:

```
run = exp.submit(tf_est)
```

Parameter server

You can also run [native distributed TensorFlow](#), which uses the parameter server model. In this method, you train across a cluster of parameter servers and workers. The workers calculate the gradients during training, while the parameter servers aggregate the gradients.

Construct the TensorFlow object:

```
from azureml.train.dnn import TensorFlow

tf_est = TensorFlow(source_directory='./my-tf-proj',
                    script_params={},
                    compute_target=compute_target,
                    entry_script='train.py',
                    node_count=2,
                    worker_count=2,
                    parameter_server_count=1,
                    distributed_backend='ps',
                    use_gpu=True)
```

Pay attention to the following parameters to the TensorFlow constructor in the above code:

PARAMETER	DESCRIPTION	DEFAULT
<code>worker_count</code>	Number of workers.	1
<code>parameter_server_count</code>	Number of parameter servers.	1
<code>distributed_backend</code>	Backend to use for distributed training. To do distributed training via parameter server, set <code>distributed_backend='ps'</code>	None

Note on `TF_CONFIG`

You will also need the network addresses and ports of the cluster for the `tf.train.ClusterSpec`, so Azure Machine Learning sets the `TF_CONFIG` environment variable for you.

The `TF_CONFIG` environment variable is a JSON string. Here is an example of the variable for a parameter server:

```
TF_CONFIG='{
  "cluster": {
    "ps": ["host0:2222", "host1:2222"],
    "worker": ["host2:2222", "host3:2222", "host4:2222"],
  },
  "task": {"type": "ps", "index": 0},
  "environment": "cloud"
}'
```

If you are using TensorFlow's high-level `tf.estimator` API, TensorFlow will parse this `TF_CONFIG` variable and build the cluster spec for you.

If you are instead using TensorFlow's lower-level core APIs for training, you need to parse the `TF_CONFIG` variable and build the `tf.train.ClusterSpec` yourself in your training code. In [this example](#), you would do so in **your training script** as follows:

```
import os, json
import tensorflow as tf

tf_config = os.environ.get('TF_CONFIG')
if not tf_config or tf_config == "":
    raise ValueError("TF_CONFIG not found.")
tf_config_json = json.loads(tf_config)
cluster_spec = tf.train.ClusterSpec(tf_config_json)
```

Once you've finished writing your training script and creating the TensorFlow object, you can submit your training job:

```
run = exp.submit(tf_est)
```

Examples

For notebooks on distributed deep learning, see:

- [how-to-use-azureml/training-with-deep-learning](#)

Learn how to run notebooks by following the article, [Use Jupyter notebooks to explore this service](#).

Next steps

- [Track run metrics during training](#)
- [Tune hyperparameters](#)
- [Deploy a trained model](#)

Tune hyperparameters for your model with Azure Machine Learning service

12/7/2018 • 12 minutes to read • [Edit Online](#)

Efficiently tune hyperparameters for your model using Azure Machine Learning service. Hyperparameter tuning includes the following steps:

- 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

What are hyperparameters?

Hyperparameters are adjustable parameters you choose to train a model that govern the training process itself. For example, to train a deep neural network, you decide the number of hidden layers in the network and the number of nodes in each layer prior to training the model. These values usually stay constant during the training process.

In deep learning / machine learning scenarios, model performance depends heavily on the hyperparameter values selected. The goal of hyperparameter exploration is to search across various hyperparameter configurations to find a configuration that results in the best performance. Typically, the hyperparameter exploration process is painstakingly manual, given that the search space is vast and evaluation of each configuration can be expensive.

Azure Machine Learning allows you to automate hyperparameter exploration in an efficient manner, saving you significant time and resources. You specify the range of hyperparameter values and a maximum number of training runs. The system then automatically launches multiple simultaneous runs with different parameter configurations and finds the configuration that results in the best performance, measured by the metric you choose. Poorly performing training runs are automatically early terminated, reducing wastage of compute resources. These resources are instead used to explore other hyperparameter configurations.

Define search space

Automatically tune hyperparameters by exploring the range of values defined for each hyperparameter.

Types of hyperparameters

Each hyperparameter can either be discrete or continuous.

Discrete hyperparameters

Discrete hyperparameters are specified as a `choice` among discrete values. `choice` can be:

- one or more comma-separated values
- a `range` object
- any arbitrary `list` object

```

{
    "batch_size": choice(16, 32, 64, 128)
    "number_of_hidden_layers": choice(range(1,5))
}

```

In this case, `batch_size` takes on one of the values [16, 32, 64, 128] and `number_of_hidden_layers` takes on one of the values [1, 2, 3, 4].

Advanced discrete hyperparameters can also be specified using a distribution. The following distributions are supported:

- `quniform(low, high, q)` - Returns a value like $\text{round}(\text{uniform}(\text{low}, \text{high}) / q) * q$
- `qloguniform(low, high, q)` - Returns a value like $\text{round}(\exp(\text{uniform}(\text{low}, \text{high})) / q) * q$
- `qnormal(mu, sigma, q)` - Returns a value like $\text{round}(\text{normal}(\mu, \sigma) / q) * q$
- `qlognormal(mu, sigma, q)` - Returns a value like $\text{round}(\exp(\text{normal}(\mu, \sigma)) / q) * q$

Continuous hyperparameters

Continuous hyperparameters are specified as a distribution over a continuous range of values. Supported distributions include:

- `uniform(low, high)` - Returns a value uniformly distributed between low and high
- `loguniform(low, high)` - Returns a value drawn according to $\exp(\text{uniform}(\text{low}, \text{high}))$ so that the logarithm of the return value is uniformly distributed
- `normal(mu, sigma)` - Returns a real value that's normally distributed with mean mu and standard deviation sigma
- `lognormal(mu, sigma)` - Returns a value drawn according to $\exp(\text{normal}(\mu, \sigma))$ so that the logarithm of the return value is normally distributed

An example of a parameter space definition:

```

{
    "learning_rate": normal(10, 3),
    "keep_probability": uniform(0.05, 0.1)
}

```

This code defines a search space with two parameters - `learning_rate` and `keep_probability`. `learning_rate` has a normal distribution with mean value 10 and a standard deviation of 3. `keep_probability` has a uniform distribution with a minimum value of 0.05 and a maximum value of 0.1.

Sampling the hyperparameter space

You can also specify the parameter sampling method to use over the hyperparameter space definition. Azure Machine Learning service supports random sampling, grid sampling, and Bayesian sampling.

Random sampling

In random sampling, hyperparameter values are randomly selected from the defined search space. Random sampling allows the search space to include both discrete and continuous hyperparameters.

```

from azureml.train.hyperdrive import RandomParameterSampling
param_sampling = RandomParameterSampling(
    "learning_rate": normal(10, 3),
    "keep_probability": uniform(0.05, 0.1),
    "batch_size": choice(16, 32, 64, 128)
)

```

Grid sampling

Grid sampling performs a simple grid search over all feasible values in the defined search space. It can only be used with hyperparameters specified using `choice`. For example, the following space has a total of six samples:

```
from azureml.train.hyperdrive import GridParameterSampling
param_sampling = GridParameterSampling( {
    "num_hidden_layers": choice(1, 2, 3),
    "batch_size": choice(16, 32)
})
```

Bayesian sampling

Bayesian sampling is based on the Bayesian optimization algorithm and makes intelligent choices on the hyperparameter values to sample next. It picks the sample based on how the previous samples performed, such that the new sample improves the reported primary metric.

When you use Bayesian sampling, the number of concurrent runs has an impact on the effectiveness of the tuning process. Typically, a smaller number of concurrent runs can lead to better sampling convergence, since the smaller degree of parallelism increases the number of runs that benefit from previously completed runs.

Bayesian sampling supports only `choice` and `uniform` distributions over the search space.

```
from azureml.train.hyperdrive import BayesianParameterSampling
param_sampling = BayesianParameterSampling( {
    "learning_rate": uniform(0.05, 0.1),
    "batch_size": choice(16, 32, 64, 128)
})
```

NOTE

Bayesian sampling does not support any early termination policy (See [Specify an early termination policy](#)). When using Bayesian parameter sampling, set `early_termination_policy = None`, or leave off the `early_termination_policy` parameter.

Specify primary metric

Specify the primary metric you want the hyperparameter tuning experiment to optimize. Each training run is evaluated for the primary metric. Poorly performing runs (where the primary metric does not meet criteria set by the early termination policy) will be terminated. In addition to the primary metric name, you also specify the goal of the optimization - whether to maximize or minimize the primary metric.

- `primary_metric_name`: The name of the primary metric to optimize. The name of the primary metric needs to exactly match the name of the metric logged by the training script. See [Log metrics for hyperparameter tuning](#).
- `primary_metric_goal`: It can be either `PrimaryMetricGoal.MAXIMIZE` or `PrimaryMetricGoal.MINIMIZE` and determines whether the primary metric will be maximized or minimized when evaluating the runs.

```
primary_metric_name="accuracy",
primary_metric_goal=PrimaryMetricGoal.MAXIMIZE
```

Optimize the runs to maximize "accuracy". Make sure to log this value in your training script.

Log metrics for hyperparameter tuning

The training script for your model must log the relevant metrics during model training. When you configure the

hyperparameter tuning, you specify the primary metric to use for evaluating run performance. (See [Specify a primary metric to optimize](#).) In your training script, you must log this metric so it is available to the hyperparameter tuning process.

Log this metric in your training script with the following sample snippet:

```
from azureml.core.run import Run
run_logger = Run.get_context()
run_logger.log("accuracy", float(val_accuracy))
```

The training script calculates the `val_accuracy` and logs it as "accuracy", which is used as the primary metric. Each time the metric is logged it is received by the hyperparameter tuning service. It is up to the model developer to determine how frequently to report this metric.

Specify early termination policy

Terminate poorly performing runs automatically with an early termination policy. Termination reduces wastage of resources and instead uses these resources for exploring other parameter configurations.

When using an early termination policy, you can configure the following parameters that control when a policy is applied:

- `evaluation_interval`: the frequency for applying the policy. Each time the training script logs the primary metric counts as one interval. Thus an `evaluation_interval` of 1 will apply the policy every time the training script reports the primary metric. An `evaluation_interval` of 2 will apply the policy every other time the training script reports the primary metric. If not specified, `evaluation_interval` is set to 1 by default.
- `delay_evaluation`: delays the first policy evaluation for a specified number of intervals. It is an optional parameter that allows all configurations to run for an initial minimum number of intervals, avoiding premature termination of training runs. If specified, the policy applies every multiple of `evaluation_interval` that is greater than or equal to `delay_evaluation`.

Azure Machine Learning service supports the following Early Termination Policies.

Bandit policy

Bandit is a termination policy based on slack factor/slack amount and evaluation interval. The policy early terminates any runs where the primary metric is not within the specified slack factor / slack amount with respect to the best performing training run. It takes the following configuration parameters:

- `slack_factor` or `slack_amount`: the slack allowed with respect to the best performing training run. `slack_factor` specifies the allowable slack as a ratio. `slack_amount` specifies the allowable slack as an absolute amount, instead of a ratio.

For example, consider a Bandit policy being applied at interval 10. Assume that the best performing run at interval 10 reported a primary metric 0.8 with a goal to maximize the primary metric. If the policy was specified with a `slack_factor` of 0.2, any training runs, whose best metric at interval 10 is less than $0.66 (0.8/(1 + slack_factor))$ will be terminated. If instead, the policy was specified with a `slack_amount` of 0.2, any training runs, whose best metric at interval 10 is less than $0.6 (0.8 - slack_amount)$ will be terminated.

- `evaluation_interval`: the frequency for applying the policy (optional parameter).
- `delay_evaluation`: delays the first policy evaluation for a specified number of intervals (optional parameter).

```
from azureml.train.hyperdrive import BanditPolicy
early_termination_policy = BanditPolicy(slack_factor = 0.1, evaluation_interval=1, delay_evaluation=5)
```

In this example, the early termination policy is applied at every interval when metrics are reported, starting at evaluation interval 5. Any run whose best metric is less than $(1/(1+0.1))$ or 91% of the best performing run will be terminated.

Median stopping policy

Median stopping is an early termination policy based on running averages of primary metrics reported by the runs. This policy computes running averages across all training runs and terminates runs whose performance is worse than the median of the running averages. This policy takes the following configuration parameters:

- `evaluation_interval` : the frequency for applying the policy (optional parameter).
- `delay_evaluation` : delays the first policy evaluation for a specified number of intervals (optional parameter).

```
from azureml.train.hyperdrive import MedianStoppingPolicy
early_termination_policy = MedianStoppingPolicy(evaluation_interval=1, delay_evaluation=5)
```

In this example, the early termination policy is applied at every interval starting at evaluation interval 5. A run will be terminated at interval 5 if its best primary metric is worse than the median of the running averages over intervals 1:5 across all training runs.

Truncation selection policy

Truncation selection cancels a given percentage of lowest performing runs at each evaluation interval. Runs are compared based on their performance on the primary metric and the lowest X% are terminated. It takes the following configuration parameters:

- `truncation_percentage` : the percentage of lowest performing runs to terminate at each evaluation interval. Specify an integer value between 1 and 99.
- `evaluation_interval` : the frequency for applying the policy (optional parameter).
- `delay_evaluation` : delays the first policy evaluation for a specified number of intervals (optional parameter).

```
from azureml.train.hyperdrive import TruncationSelectionPolicy
early_termination_policy = TruncationSelectionPolicy(evaluation_interval=1, truncation_percentage=20,
delay_evaluation=5)
```

In this example, the early termination policy is applied at every interval starting at evaluation interval 5. A run will be terminated at interval 5, if its performance at interval 5 is in the lowest 20% of performance of all runs at interval 5.

No termination policy

If you want all training runs to run to completion, set policy to None. This will have the effect of not applying any early termination policy.

```
policy=None
```

Default policy

If no policy is specified, the hyperparameter tuning service will let all training runs run to completion.

NOTE

If you are looking for a conservative policy that provides savings without terminating promising jobs, you can use a Median Stopping Policy with `evaluation_interval` 1 and `delay_evaluation` 5. These are conservative settings, that can provide approximately 25%-35% savings with no loss on primary metric (based on our evaluation data).

Allocate resources

Control your resource budget for your hyperparameter tuning experiment by specifying the maximum total number of training runs. Optionally specify the maximum duration for your hyperparameter tuning experiment.

- `max_total_runs` : Maximum total number of training runs that will be created. Upper bound - there may be fewer runs, for instance, if the hyperparameter space is finite and has fewer samples. Must be a number between 1 and 1000.
- `max_duration_minutes` : Maximum duration in minutes of the hyperparameter tuning experiment. Parameter is optional, and if present, any runs that would be running after this duration are automatically canceled.

NOTE

If both `max_total_runs` and `max_duration_minutes` are specified, the hyperparameter tuning experiment terminates when the first of these two thresholds is reached.

Additionally, specify the maximum number of training runs to run concurrently during your hyperparameter tuning search.

- `max_concurrent_runs` : Maximum number of runs to run concurrently at any given moment. If none specified, all `max_total_runs` will be launched in parallel. If specified, must be a number between 1 and 100.

NOTE

The number of concurrent runs is gated on the resources available in the specified compute target. Hence, you need to ensure that the compute target has the available resources for the desired concurrency.

Allocate resources for hyperparameter tuning:

```
max_total_runs=20,  
max_concurrent_runs=4
```

This code configures the hyperparameter tuning experiment to use a maximum of 20 total runs, running 4 configurations at a time.

Configure experiment

Configure your hyperparameter tuning experiment using the defined hyperparameter search space, early termination policy, primary metric, and resource allocation from the sections above. Additionally, provide an `estimator` that will be called with the sampled hyperparameters. The `estimator` describes the training script you run, the resources per job (single or multi-gpu), and the compute target to use. Since concurrency for your hyperparameter tuning experiment is gated on the resources available, ensure that the compute target specified in the `estimator` has sufficient resources for your desired concurrency. (For more information on estimators, see [how to train models](#).)

Configure your hyperparameter tuning experiment:

```
from azureml.train.hyperdrive import HyperDriveRunConfig
hyperdrive_run_config = HyperDriveRunConfig(estimator=estimator,
                                             hyperparameter_sampling=param_sampling,
                                             policy=early_termination_policy,
                                             primary_metric_name="accuracy",
                                             primary_metric_goal=PrimaryMetricGoal.MAXIMIZE,
                                             max_total_runs=100,
                                             max_concurrent_runs=4)
```

Submit experiment

Once you define your hyperparameter tuning configuration, submit an experiment:

```
from azureml.core.experiment import Experiment
experiment = Experiment(workspace, experiment_name)
hyperdrive_run = experiment.submit(hyperdrive_run_config)
```

`experiment_name` is the name you assign to your hyperparameter tuning experiment, and `workspace` is the workspace in which you want to create the experiment (For more information on experiments, see [How does Azure Machine Learning service work?](#))

Visualize experiment

The Azure Machine Learning SDK provides a Notebook widget that visualizes the progress of your training runs. The following snippet visualizes all your hyperparameter tuning runs in one place in a Jupyter notebook:

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

This code displays a table with details about the training runs for each of the hyperparameter configurations.

Run Properties	
Status	Completed
Start Time	9/4/2018 2:55:54 PM
Duration	7 days, 0:01:41
Run Id	cifar_1536098154351
Max concurrent runs	4
Max total runs	100

Output Logs

[2018-09-11T21:57:36.389083][CONTROLLER][INFO]Experiment was 'ExperimentStatus.RUNNING', is 'ExperimentStatus.FINISHED'.

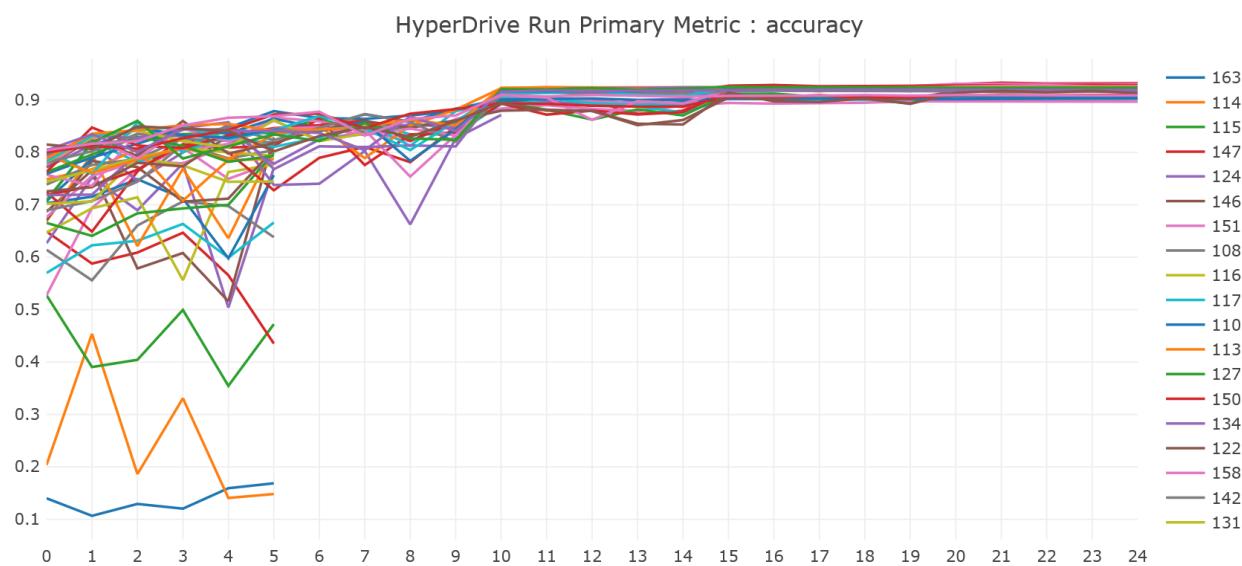
Run is completed.

Failed (6)	Completed (36)			Canceled (58)		
Run	Best Metric*	Status	Started	Duration	Run Id	
103	0.909600019454956	Completed	Sep 4, 2018 2:56 PM	5:58:16	swatig-cifar_1536098154351_0	
105	0.9283999800682068	Completed	Sep 4, 2018 2:56 PM	6:04:28	swatig-cifar_1536098154351_3	
104	0.9247000217437744	Completed	Sep 4, 2018 2:56 PM	3:32:35	swatig-cifar_1536098154351_2	
106	0.9251000285148621	Completed	Sep 4, 2018 2:56 PM	6:01:17	swatig-cifar_1536098154351_1	
107	0.9108999967575073	Completed	Sep 4, 2018 6:29 PM	4:18:35	swatig-cifar_1536098154351_4	

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

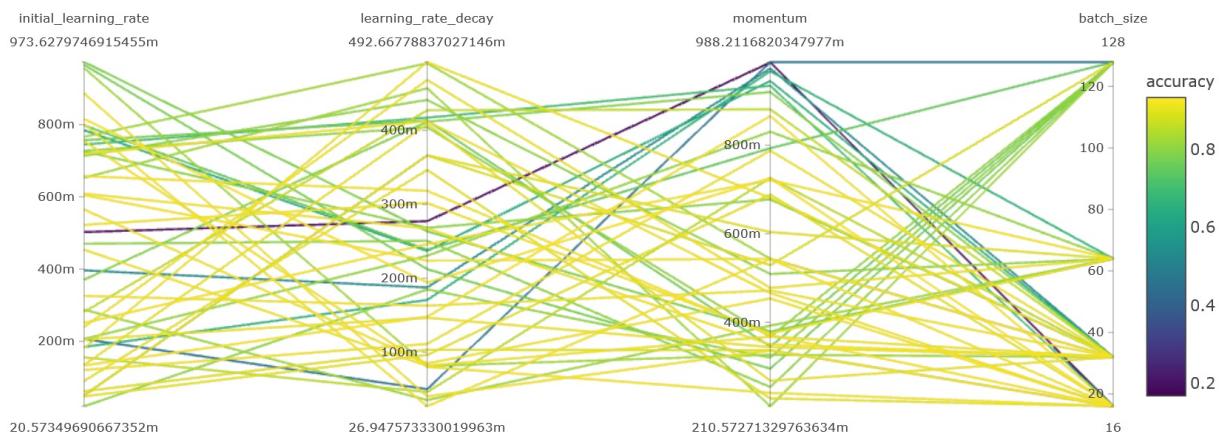
* The best metric field is obtained from the min/max of primary metric achieved by a run

You can also visualize the performance of each of the runs as training progresses.

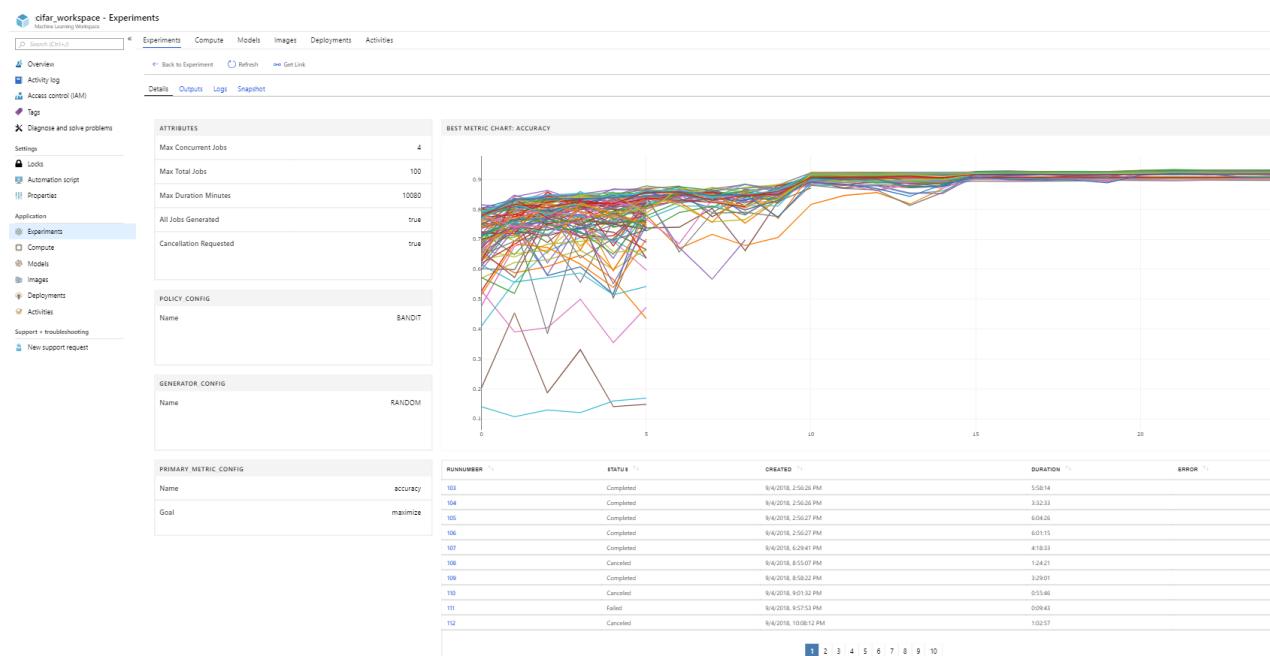


Additionally, you can visually identify the correlation between performance and values of individual hyperparameters using a Parallel Coordinates Plot.

Parallel Coordinates Chart



You can visualize all your hyperparameter tuning runs in the Azure web portal as well. For more information on how to view an experiment in the web portal, see [how to track experiments](#).



Find the best model

Once all of the hyperparameter tuning runs have completed, identify the best performing configuration and the corresponding hyperparameter values:

```
best_run = hyperdrive_run.get_best_run_by_primary_metric()
best_run_metrics = best_run.get_metrics()
parameter_values = best_run.get_details()['runDefinition']['Arguments']

print('Best Run Id: ', best_run.id)
print('\n Accuracy:', best_run_metrics['accuracy'])
print('\n learning rate:', parameter_values[3])
print('\n keep probability:', parameter_values[5])
print('\n batch size:', parameter_values[7])
```

Sample notebook

Refer to these notebooks:

- [how-to-use-azureml/training-with-deep-learning/train-hyperparameter-tune-deploy-with-pytorch](#)
- [how-to-use-azureml/training-with-deep-learning/train-hyperparameter-tune-deploy-with-tensorflow](#)

Learn how to run notebooks by following the article, [Use Jupyter notebooks to explore this service](#).

Next steps

- [Track an experiment](#)
- [Deploy a trained model](#)

Track experiments and training metrics in Azure Machine Learning

12/10/2018 • 14 minutes to read • [Edit Online](#)

In the Azure Machine Learning service, you can track your experiments and monitor metrics to enhance the model creation process. In this article, you'll learn about the different ways to add logging to your training script, how to submit the experiment with **start_logging** and **ScriptRunConfig**, how to check the progress of a running job, and how to view the results of a run.

List of training metrics

The following metrics can be added to a run while training an experiment. To view a more detailed list of what can be tracked on a run, see the [Run class reference documentation](#).

Type	Python Function	Notes
Scalar values	Function: <pre>run.log(name, value, description='')</pre> Example: <pre>run.log("accuracy", 0.95)</pre>	Log a numerical or string value to the run with the given name. Logging a metric to a run causes that metric to be stored in the run record in the experiment. You can log the same metric multiple times within a run, the result being considered a vector of that metric.
Lists	Function: <pre>run.log_list(name, value, description='')</pre> Example: <pre>run.log_list("accuracies", [0.6, 0.7, 0.87])</pre>	Log a list of values to the run with the given name.
Row	Function: <pre>'run.log_row(name, description=None, **kwargs)</pre> Example: <pre>run.log_row("Y over X", x=1, y=0.4)</pre>	Using <i>log_row</i> creates a metric with multiple columns as described in <i>kwargs</i> . Each named parameter generates a column with the value specified. <i>log_row</i> can be called once to log an arbitrary tuple, or multiple times in a loop to generate a complete table.
Table	Function: <pre>run.log_table(name, value, description='')</pre> Example: <pre>run.log_table("Y over X", {"x":[1, 2, 3], "y":[0.6, 0.7, 0.89]})</pre>	Log a dictionary object to the run with the given name.
Images	Function: <pre>run.log_image(name, path=None, plot=None)</pre> Example: <pre>run.log_image("ROC", plt)</pre>	Log an image to the run record. Use <i>log_image</i> to log an image file or a matplotlib plot to the run. These images will be visible and comparable in the run record.

TYPE	PYTHON FUNCTION	NOTES
Tag a run	<p>Function:</p> <pre>run.tag(key, value=None)</pre> <p>Example:</p> <pre>run.tag("selected", "yes")</pre>	Tag the run with a string key and optional string value.
Upload file or directory	<p>Function:</p> <pre>run.upload_file(name, path_or_stream)</pre> <p>Example:</p> <pre>run.upload_file("best_model.pkl", "./model.pkl")</pre>	Upload a file to the run record. Runs automatically capture file in the specified output directory, which defaults to "./outputs" for most run types. Use <code>upload_file</code> only when additional files need to be uploaded or an output directory is not specified. We suggest adding <code>outputs</code> to the name so that it gets uploaded to the outputs directory. You can list all of the files that are associated with this run record by called <code>run.get_file_names()</code>

NOTE

Metrics for scalars, lists, rows, and tables can have type: float, integer, or string.

Start logging metrics

If you want to track or monitor your experiment, you must add code to start logging when you submit the run. The following are ways to trigger the run submission:

- **Run.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.
- **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.

Set up the workspace

Before adding logging and submitting an experiment, you must set up the workspace.

1. Load the workspace. To learn more about setting the workspace configuration, follow the [quickstart](#).

```
from azureml.core import Experiment, Run, Workspace
import azureml.core

ws = Workspace(workspace_name = <<workspace_name>>,
               subscription_id = <<subscription_id>>,
               resource_group = <<resource_group>>)
```

Option 1: Use start_logging

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.

The following example trains a simple sklearn Ridge model locally in a local Jupyter notebook. To learn more

about submitting experiments to different environments, see [Set up compute targets for model training with Azure Machine Learning service](#).

1. Create a training script in a local Jupyter notebook.

```
# load diabetes dataset, a well-known small dataset that comes with scikit-learn
from sklearn.datasets import load_diabetes
from sklearn.linear_model import Ridge
from sklearn.metrics import mean_squared_error
from sklearn.model_selection import train_test_split
from sklearn.externals import joblib

X, y = load_diabetes(return_X_y = True)
columns = ['age', 'gender', 'bmi', 'bp', 's1', 's2', 's3', 's4', 's5', 's6']
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.2, random_state = 0)
data = {
    "train": {"X": X_train, "y": y_train},
    "test": {"X": X_test, "y": y_test}
}
reg = Ridge(alpha = 0.03)
reg.fit(data['train']['X'], data['train']['y'])
preds = reg.predict(data['test']['X'])
print('Mean Squared Error is', mean_squared_error(preds, data['test']['y']))
joblib.dump(value = reg, filename = 'model.pkl');
```

2. Add experiment tracking using the Azure Machine Learning service SDK, and upload a persisted model into the experiment run record. The following code adds tags, logs, and uploads a model file to the experiment run.

```
# Get an experiment object from Azure Machine Learning
experiment = Experiment(workspace = ws, name = "train-within-notebook")

# Create a run object in the experiment
run = experiment.start_logging()# Log the algorithm parameter alpha to the run
run.log('alpha', 0.03)

# Create, fit, and test the scikit-learn Ridge regression model
regression_model = Ridge(alpha=0.03)
regression_model.fit(data['train']['X'], data['train']['y'])
preds = regression_model.predict(data['test']['X'])

# Output the Mean Squared Error to the notebook and to the run
print('Mean Squared Error is', mean_squared_error(data['test']['y'], preds))
run.log('mse', mean_squared_error(data['test']['y'], preds))

# Save the model to the outputs directory for capture
joblib.dump(value=regression_model, filename='outputs/model.pkl')

# Take a snapshot of the directory containing this notebook
run.take_snapshot('./')

# Complete the run
run.complete()
```

The script ends with `run.complete()`, which marks the run as completed. This function is typically used in interactive notebook scenarios.

Option 2: Use ScriptRunConfig

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.

This example expands on the basic sklearn Ridge model from above. It does a simple parameter sweep to sweep over alpha values of the model to capture metrics and trained models in runs under the experiment. The example runs locally against a user-managed environment.

1. Create a training script. This code uses `%%writefile%%` to write the training code out to the script folder as

```
train.py .
```

```
%%writefile $project_folder/train.py

import os
from sklearn.datasets import load_diabetes
from sklearn.linear_model import Ridge
from sklearn.metrics import mean_squared_error
from sklearn.model_selection import train_test_split
from azureml.core.run import Run
from sklearn.externals import joblib

import numpy as np

#os.makedirs('./outputs', exist_ok = True)

X, y = load_diabetes(return_X_y = True)

run = Run.get_context()

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.2, random_state = 0)
data = {"train": {"X": X_train, "y": y_train},
        "test": {"X": X_test, "y": y_test}}

# list of numbers from 0.0 to 1.0 with a 0.05 interval
alphas = mylib.get_alphas()

for alpha in alphas:
    # Use Ridge algorithm to create a regression model
    reg = Ridge(alpha = alpha)
    reg.fit(data["train"]["X"], data["train"]["y"])

    preds = reg.predict(data["test"]["X"])
    mse = mean_squared_error(preds, data["test"]["y"])
    # log the alpha and mse values
    run.log('alpha', alpha)
    run.log('mse', mse)

    model_file_name = 'ridge_{0:.2f}.pk1'.format(alpha)
    # save model in the outputs folder so it automatically get uploaded
    with open(model_file_name, "wb") as file:
        joblib.dump(value = reg, filename = model_file_name)

    # upload the model file explicitly into artifacts
    run.upload_file(name = model_file_name, path_or_stream = model_file_name)

    # register the model
    #run.register_model(file_name = model_file_name)

    print('alpha is {0:.2f}, and mse is {1:.2f}'.format(alpha, mse))
```

2. The `train.py` script references `mylib.py`. This file allows you to get the list of alpha values to use in the ridge model.

```
%>writefile $script_folder/mylib.py
import numpy as np

def get_alphas():
    # list of numbers from 0.0 to 1.0 with a 0.05 interval
    return np.arange(0.0, 1.0, 0.05)
```

3. Configure a user-managed local environment.

```
from azureml.core.runconfig import RunConfiguration

# Editing a run configuration property on-fly.
run_config_user_managed = RunConfiguration()

run_config_user_managed.environment.python.user_managed_dependencies = True

# You can choose a specific Python environment by pointing to a Python path
#run_config.environment.python.interpreter_path = '/home/user/miniconda3/envs/sdk2/bin/python'
```

4. Submit the `train.py` script to run in the user-managed environment. This whole script folder is submitted for training, including the `mylib.py` file.

```
from azureml.core import ScriptRunConfig

experiment = Experiment(workspace=ws, name="train-on-local")
src = ScriptRunConfig(source_directory = '.', script = 'train.py', run_config =
run_config_user_managed)
run = experiment.submit(src)
```

View run details

Monitor run with Jupyter notebook widgets

When you use the **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.

1. View the Jupyter widget while waiting for the run to complete.

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

Edit Metadata

```

1 from azureml.train.widgets import RunDetails
2 RunDetails(run).show()

```

Run Properties		Output Logs
Status	Running	Uploading experiment status to history service. Adding run profile attachment azureml-logs/80_driver_log.txt
Start Time	9/15/2018 7:15:37 PM	
Duration	0:00:20	
Run Id	train-on-local_1537053337_839d0780	
Arguments	N/A	

alpha

mse

The chart shows the relationship between alpha (x-axis, 0 to 7) and mse (y-axis, 3300 to 3400). As alpha increases, mse decreases.

alpha	mse
0.00	3424.32
0.05	3408.92
0.10	3372.65
0.15	3345.15
0.20	3325.29
0.25	3311.56
0.30	3302.67

[Click here to see the run in Azure portal](#)

2. [For automated machine learning runs] To access the charts from a previous run. Please replace <><experiment_name>> with the appropriate experiment name:

```

from azureml.train.widgets import RunDetails
from azureml.core.run import Run

experiment = Experiment(workspace, <><experiment_name>>)
run_id = 'autoML_my_runID' #replace with run_ID
run = Run(experiment, run_id)
RunDetails(run).show()

```

AutoML_ed181129-3876-452a-82b0-39f33a8290b7:
Status: **Completed**

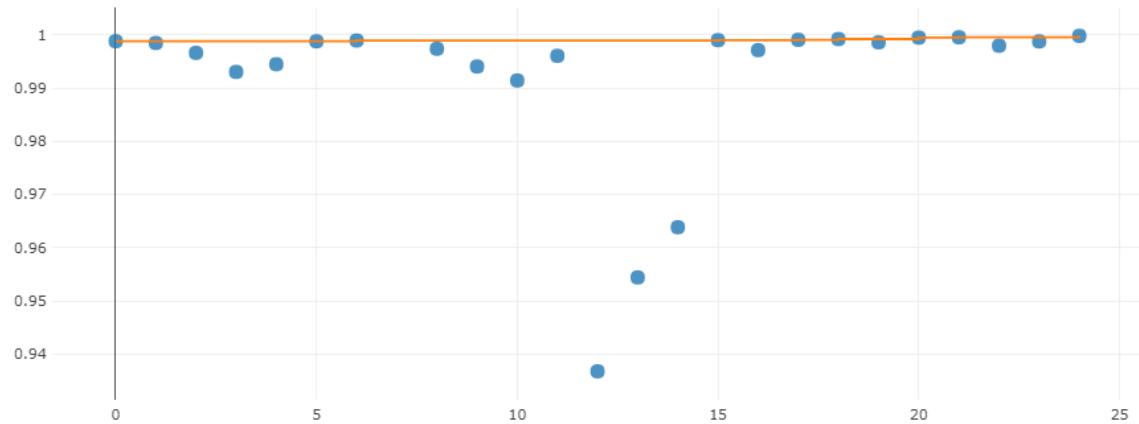
Status -	0	2	4	6	8	10	12	14	16	18	20	22	24

Iteration	Pipeline	Iteration metric	Best metric	Status	Duration	Started
0	StandardScalerWrapper, KNN	0.99883057	0.99883057	Completed	0:00:00	Nov 26, 2018 1:30 PM
1	StandardScalerWrapper, KNN	0.99849239	0.99883057	Completed	0:00:00	Nov 26, 2018 1:30 PM
2	MaxAbsScaler, LightGBM	0.99664006	0.99883057	Completed	0:00:00	Nov 26, 2018 1:30 PM
3	StandardScalerWrapper, LightGBM	0.9930566	0.99883057	Completed	0:00:00	Nov 26, 2018 1:31 PM
4	StandardScalerWrapper, LogisticRegression	0.9944965	0.99883057	Completed	0:00:00	Nov 26, 2018 1:31 PM

Pages: 1 2 3 4 5 Next Last 5 per page

AUC_weighted

Run with metric : AUC_weighted



[Click here to see the run in Azure portal](#)

To view further details of a pipeline click on the Pipeline you would like to explore in the table, and the charts will render in a pop-up from the Azure portal.

Get log results upon completion

Model training and monitoring occur in the background so that you can run other tasks while you wait. You can also wait until the model has completed training before running more code. When you use **ScriptRunConfig**, you can use `run.wait_for_completion(show_output = True)` to show when the model training is complete. The `show_output` flag gives you verbose output.

Query run metrics

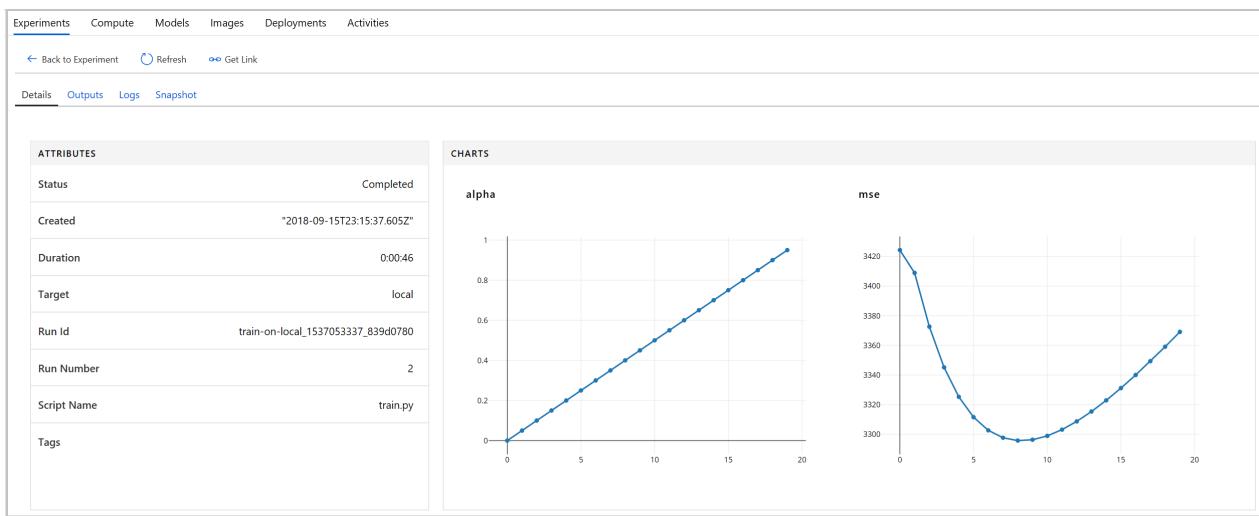
You can view the metrics of a trained model using `run.get_metrics()`. You can now get all of the metrics that were logged in the example above to determine the best model.

View the experiment in the Azure portal

When an experiment has finished running, you can browse to the recorded experiment run record. You can do access the history in two ways:

- Get the URL to the run directly `print(run.get_portal_url())`
- View the run details by submitting the name of the run (in this case, `run`). This way points you to the experiment name, ID, type, status, details page, a link to the Azure portal, and a link to documentation.

The link for the run brings you directly to the run details page in the Azure portal. Here you can see any properties, tracked metrics, images, and charts that are logged in the experiment. In this case, we logged MSE and the alpha values.



You can also view any outputs or logs for the run, or download the snapshot of the experiment you submitted so you can share the experiment folder with others.

Viewing charts in run details

There are various ways to use the logging APIs to record different types of metrics during a run and view them as charts in the Azure portal.

LOGGED VALUE	EXAMPLE CODE	VIEW IN PORTAL
Log an array of numeric values	<pre>run.log_list(name='Fibonacci', value=[0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89])</pre>	single-variable line chart
Log a single numeric value with the same metric name repeatedly used (like from within a for loop)	<pre>for i in tqdm(range(-10, 10)): run.log(name='Sigmoid', value=1 / (1 + np.exp(-i))) angle = i / 2.0</pre>	Single-variable line chart
Log a row with 2 numerical columns repeatedly	<pre>run.log_row(name='Cosine Wave', angle=angle, cos=np.cos(angle)) sines['angle'].append(angle) sines['sine'].append(np.sin(angle))</pre>	Two-variable line chart
Log table with 2 numerical columns	<pre>run.log_table(name='Sine Wave', value=sines)</pre>	Two-variable line chart

Understanding automated ML charts

After submitting an automated ML job in a notebook, a history of these runs can be found in your machine learning service workspace.

Learn more about:

- [Charts and curves for classification models](#)
- [Charts and graphs for regression models](#)
- [Model explainability](#)

View the run charts

1. Go to your workspace.
2. Select **Experiments** in the leftmost panel of your workspace.

The screenshot shows the Azure Machine Learning studio interface. On the left, there's a navigation sidebar with sections like Overview, Activity log, Access control (IAM), Tags, Diagnose and solve problems, Settings, Locks, Automation script, Properties, Application, and Experiments. The 'Experiments' link is circled in red.

3. Select the experiment you are interested in.

Experiments

Refresh

NAME ↑↓	CREATED ↑↓
energydemandforecasting	10/31/2018, 10:44:51 AM
local-classification_1016	10/16/2018, 11:02:25 AM
local-regression_ss_Demo	10/9/2018, 8:37:07 AM
local-regression	10/1/2018, 1:07:41 PM
local-classification	10/1/2018, 1:07:03 PM
local-missing-data	10/1/2018, 1:02:18 PM

4. In the table, select the Run Number.



5. In the table, select the Iteration Number for the model that you would like to explore further.

Iterations							
Iteration	Run_Preprocessor	Run_Algorithm	Normalized_Root_Mean_Squared_Error	Status	Created	Duration	
5	SparseNormalizer	DecisionTreeRegressor	0.06312059746871934	Completed	10/31/2018, 10:47:23 AM	23:58:34	
2	MaxAbsScaler	RandomForestRegressor	0.07021324452854548	Completed	10/31/2018, 10:46:00 AM	23:58:37	
1	SparseNormalizer	DecisionTreeRegressor	0.08182806797817412	Completed	10/31/2018, 10:45:33 AM	23:58:35	
9	StandardScalerWrapper	LassoLars	0.10910856865252272	Completed	10/31/2018, 10:49:11 AM	23:58:34	
7	StandardScalerWrapper	ElasticNet	0.10984011331262775	Completed	10/31/2018, 10:48:18 AM	23:58:33	
6	StandardScalerWrapper	ElasticNet	0.10988317343439677	Completed	10/31/2018, 10:47:50 AM	23:58:35	
4	MaxAbsScaler	ElasticNet	0.13711681172618576	Completed	10/31/2018, 10:46:56 AM	23:58:34	
3	SparseNormalizer	DecisionTreeRegressor	0.13961105249753472	Completed	10/31/2018, 10:46:30 AM	23:58:33	
8	StandardScalerWrapper	ElasticNet	0.14459679782725773	Completed	10/31/2018, 10:48:44 AM	23:58:34	
0	TruncatedSVDWrapper	DecisionTreeRegressor	0.14460494174552974	Completed	10/31/2018, 10:45:05 AM	23:58:35	

Classification

For every classification model that you build by using the automated machine learning capabilities of Azure Machine Learning, you can see the following charts:

- [Confusion matrix](#)
- [Precision-Recall chart](#)
- [Receiver operating characteristics \(or ROC\)](#)
- [Lift curve](#)
- [Gains curve](#)
- [Calibration plot](#)

Confusion matrix

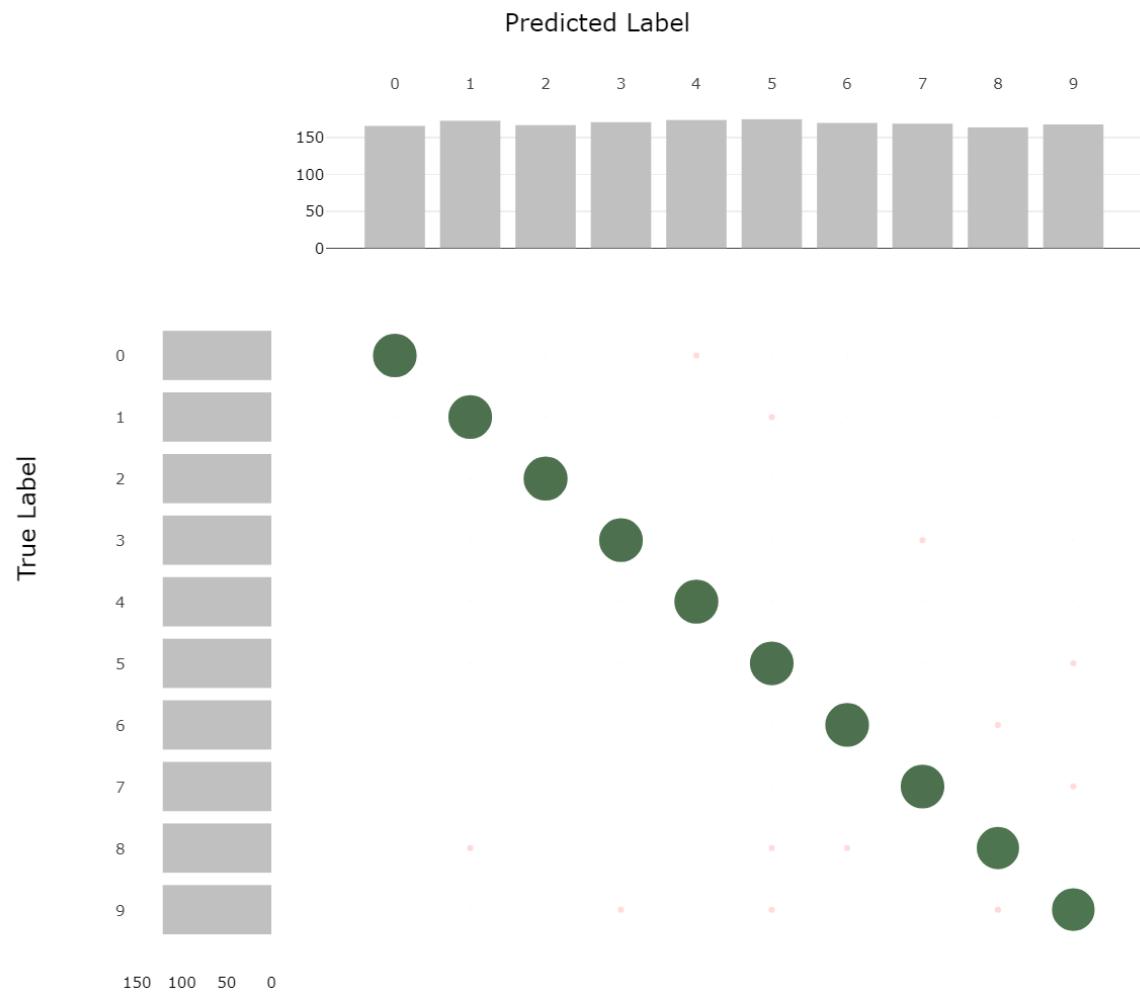
A confusion matrix is used to describe the performance of a classification model. Each row displays the instances of the true class, and each column represents the instances of the predicted class. The confusion matrix shows the correctly classified labels and the incorrectly classified labels for a given model.

For classification problems, Azure Machine Learning automatically provides a confusion matrix for each model that is built. For each confusion matrix, automated ML will show the correctly classified labels as green, and incorrectly classified labels as red. The size of the circle represents the number of samples in that bin. In addition, the frequency count of each predicted label and each true label is provided in the adjacent bar charts.

Example 1: A classification model with poor accuracy



Example 2: A classification model with high accuracy (ideal)

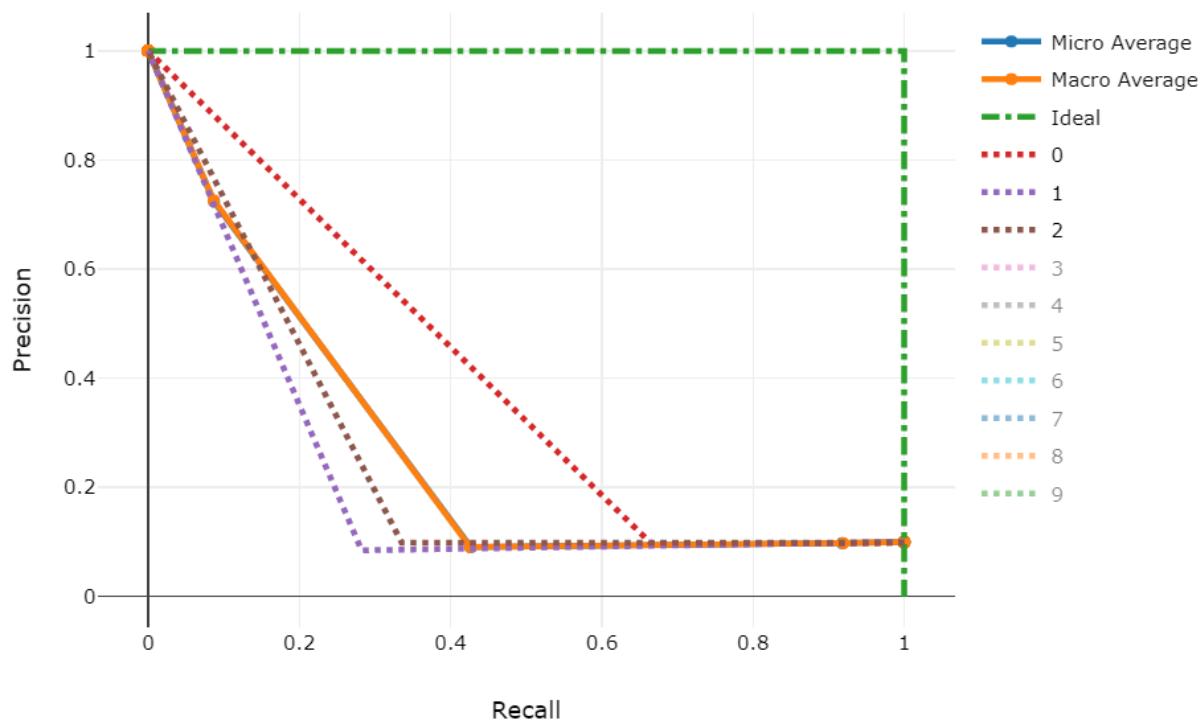


Precision-recall chart

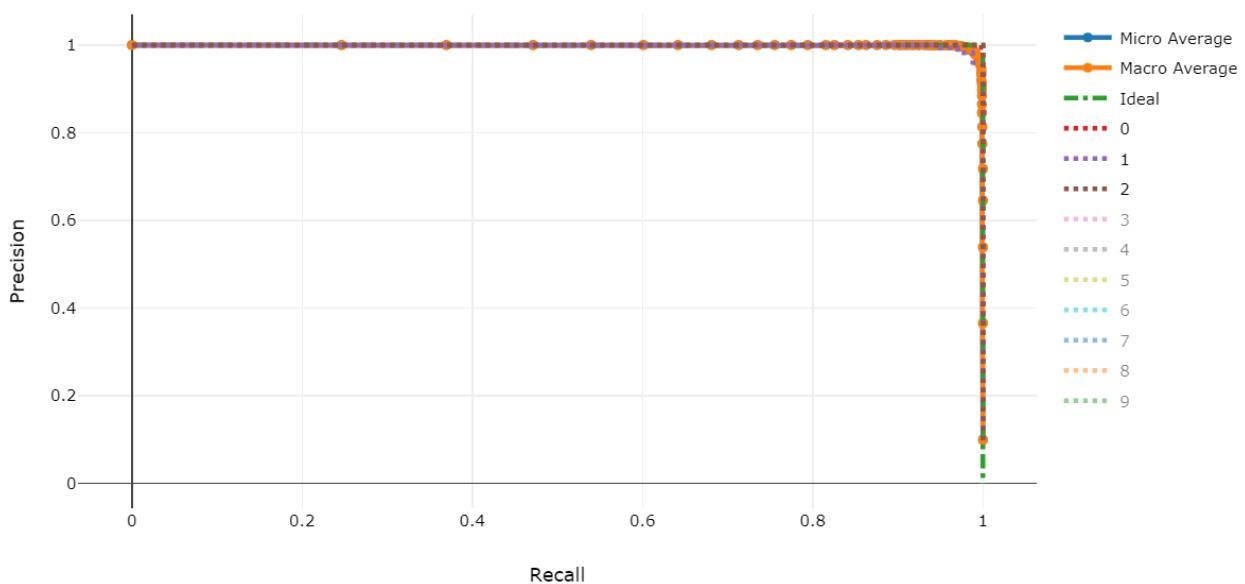
With this chart, you can compare the precision-recall curves for each model to determine which model has an acceptable relationship between precision and recall for your particular business problem. This chart shows Macro Average Precision-Recall, Micro Average Precision-Recall, and the precision-recall associated with all classes for a model.

The term Precision represents that ability for a classifier to label all instances correctly. Recall represents the ability for a classifier to find all instances of a particular label. The precision-recall curve shows the relationship between these two concepts. Ideally, the model would have 100% precision and 100% accuracy.

Example 1: A classification model with low precision and low recall



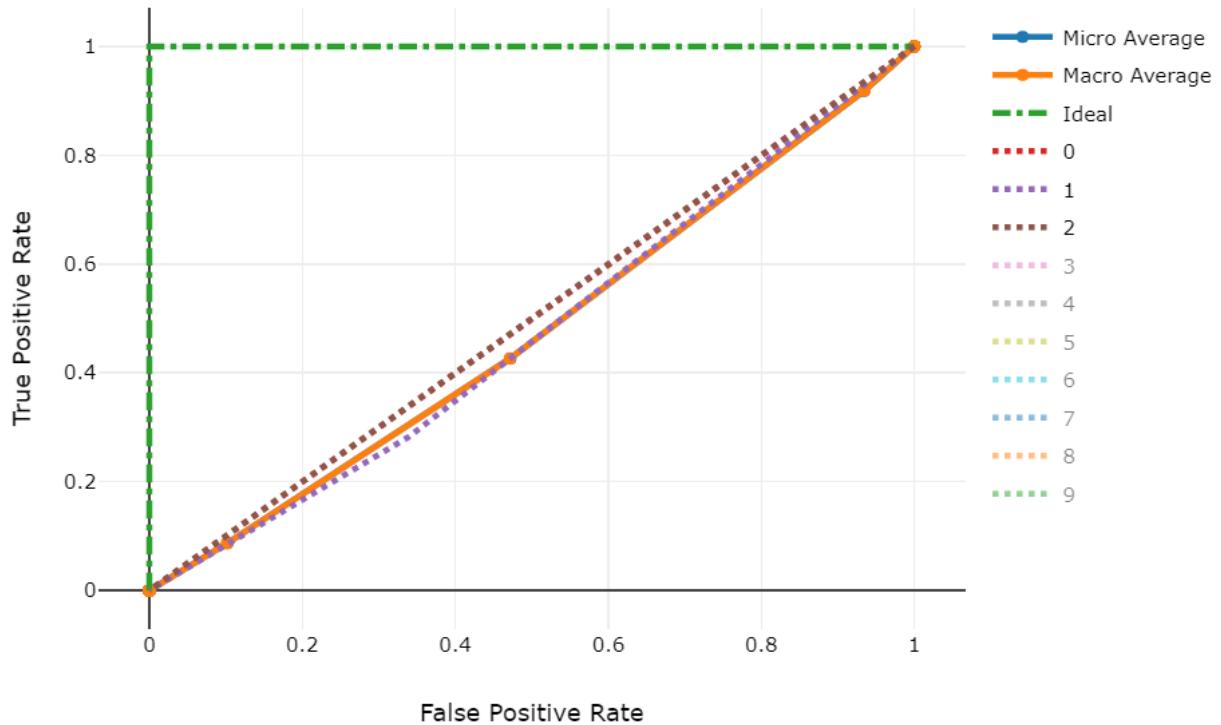
Example 2: A classification model with ~100% precision and ~100% recall (ideal)



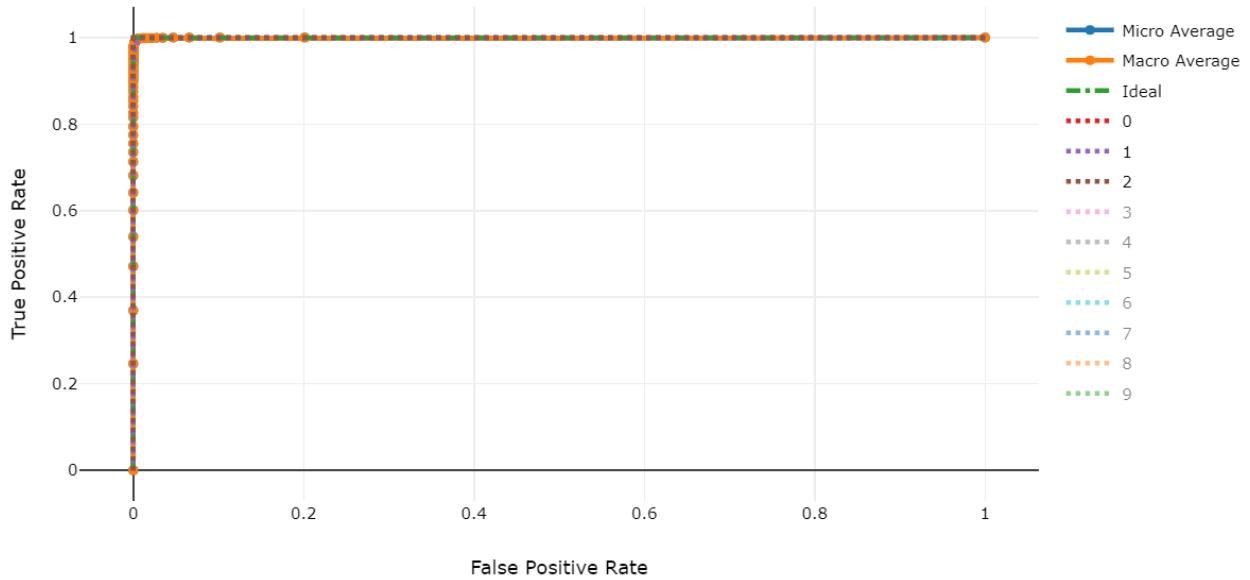
ROC

Receiver operating characteristic (or ROC) is a plot of the correctly classified labels vs. the incorrectly classified labels for a particular model. The ROC curve can be less informative when training models on datasets with high bias, as it will not show the false positive labels.

Example 1: A classification model with low true labels and high false labels



Example 2: A classification model with high true labels and low false labels

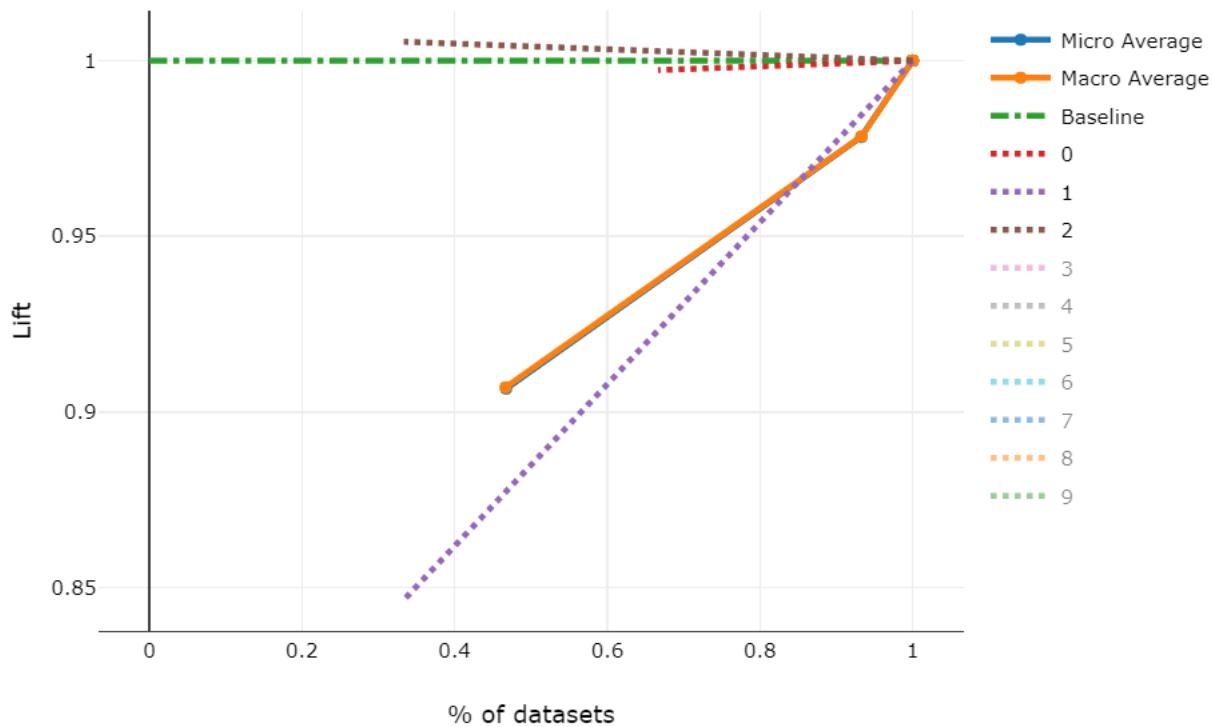


Lift curve

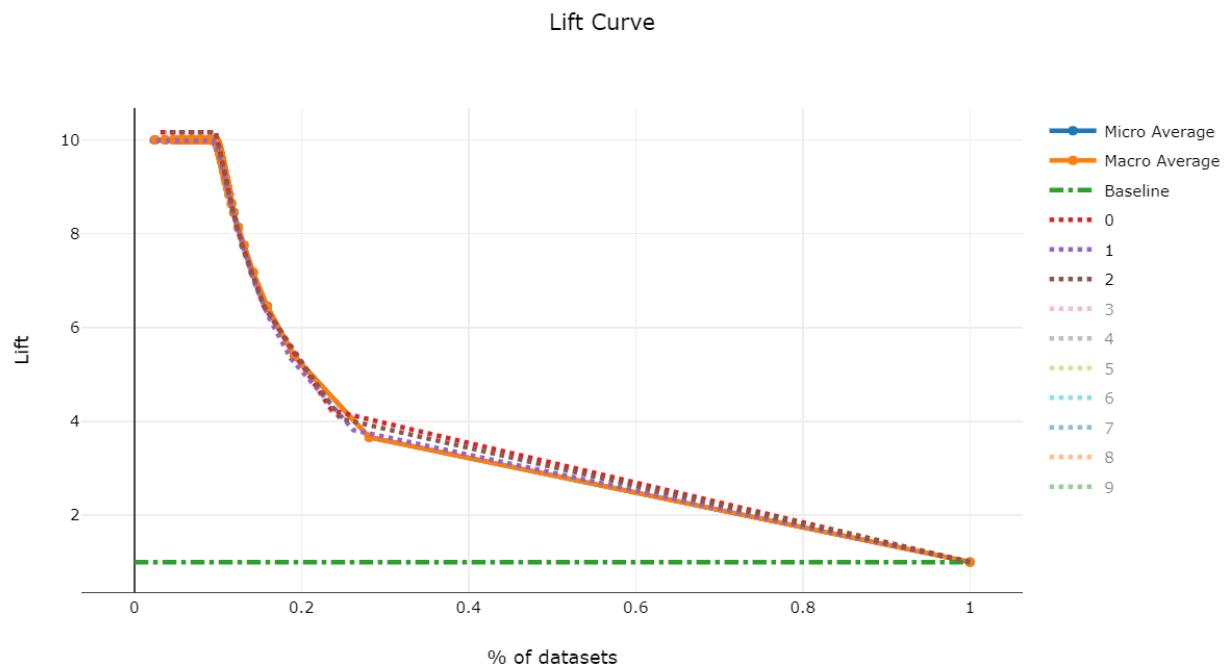
You can compare the lift of the model built automatically with Azure Machine Learning to the baseline in order to view the value gain of that particular model.

Lift charts are used to evaluate the performance of a classification model. It shows how much better you can expect to do with a model compared to without a model.

Example 1: Model performs worse than a random selection model



Example 2: Model performs better than a random selection model

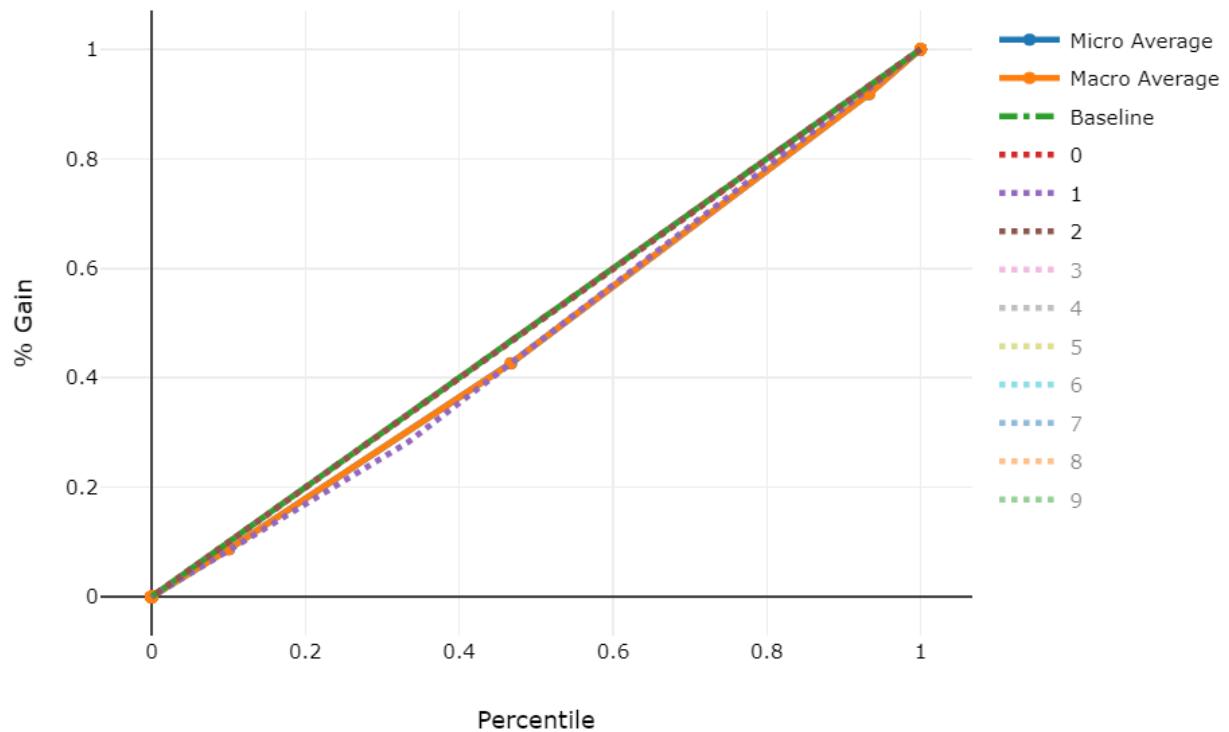


Gains curve

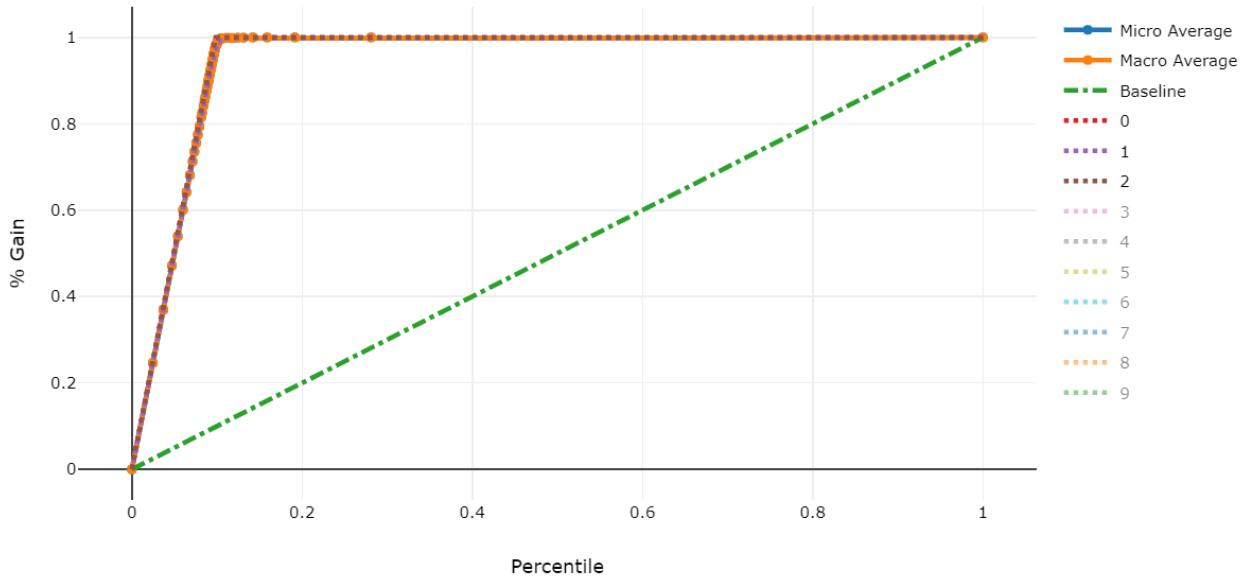
A gains chart evaluates the performance of a classification model by each portion of the data. It shows for each percentile of the data set, how much better you can expect to perform compared against a random selection model.

Use the cumulative gains chart to help you choose the classification cutoff using a percentage that corresponds to a desired gain from the model. This information provides another way of looking at the results in the accompanying lift chart.

Example 1: A classification model with minimal gain



Example 2: A classification model with significant gain

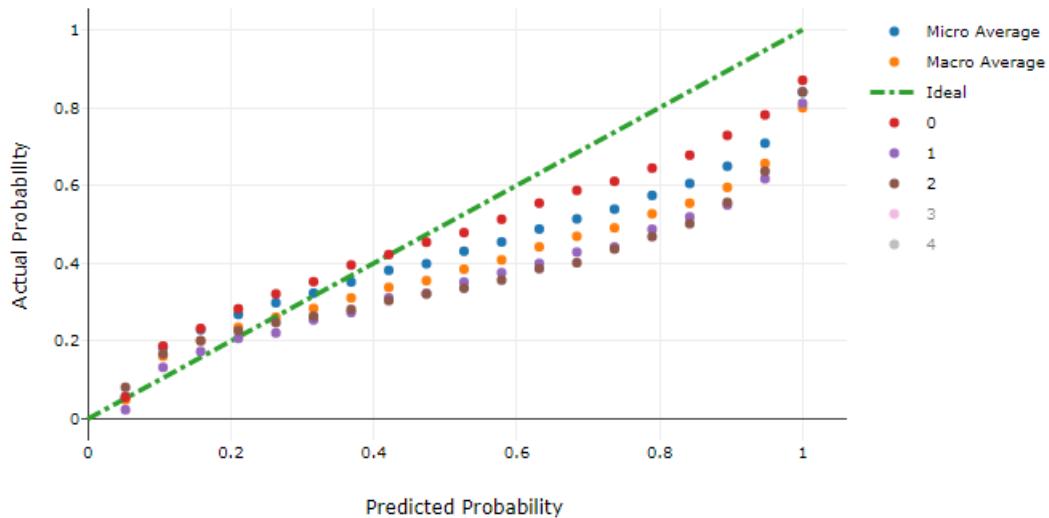


Calibration plot

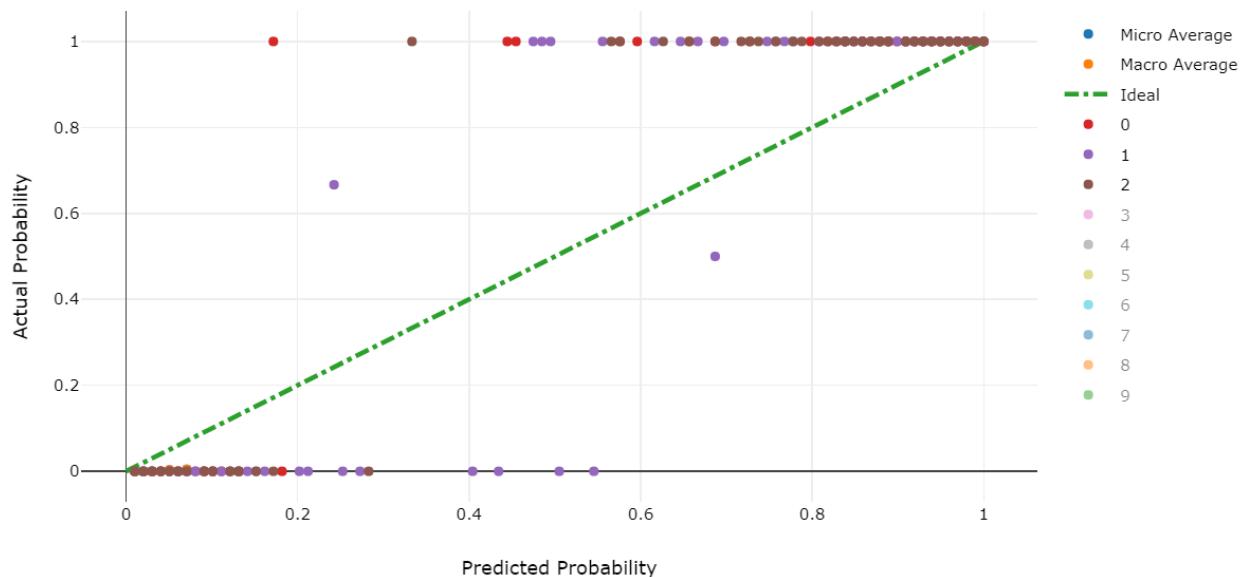
For all classification problems, you can review the calibration line for micro-average, macro-average, and each class in a given predictive model.

A calibration plot is used to display the confidence of a predictive model. It does this by showing the relationship between the predicted probability and the actual probability, where "probability" represents the likelihood that a particular instance belongs under some label. A well calibrated model aligns with the $y=x$ line, where it is reasonably confident in its predictions. An over-confident model aligns with the $y=0$ line, where the predicted probability is present but there is no actual probability.

Example 1: A more well-calibrated model



Example 2: An over-confident model



Regression

For every regression model, you build using the automated machine learning capabilities of Azure Machine Learning, you can see the following charts:

- [Predicted vs. True](#)
- [Histogram of residuals](#)

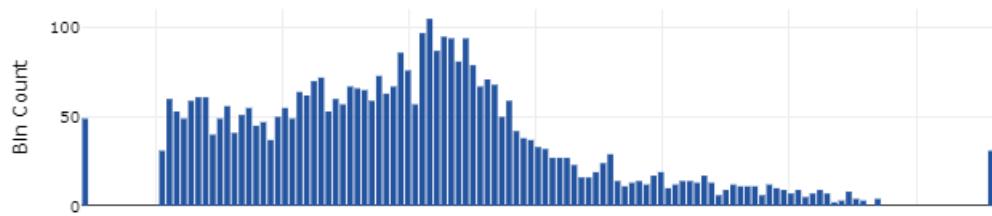
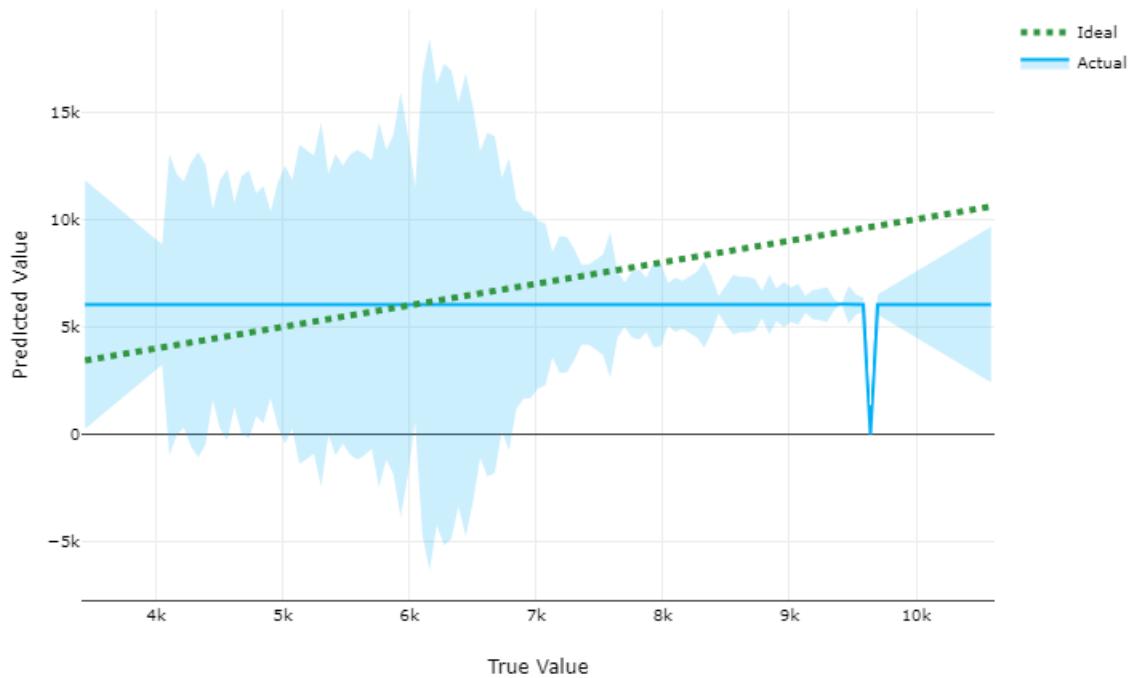
Predicted vs. True

Predicted vs. True shows the relationship between a predicted value and its correlating true value for a regression problem. This graph can be used to measure performance of a model as the closer to the $y=x$ line the predicted values are, the better the accuracy of a predictive model.

After each run, you can see a predicted vs. true graph for each regression model. To protect data privacy, values are binned together and the size of each bin is shown as a bar graph on the bottom portion of the chart area. You can compare the predictive model, with the lighter shade area showing error margins, against the ideal value of where the model should be.

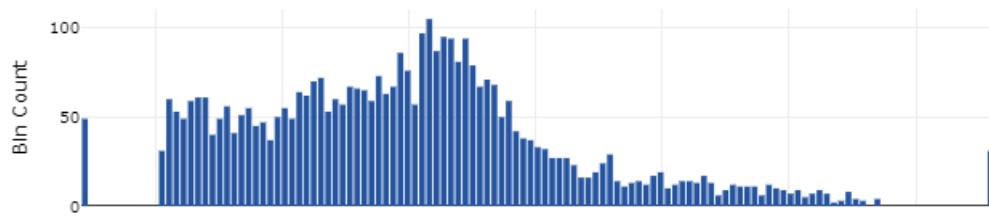
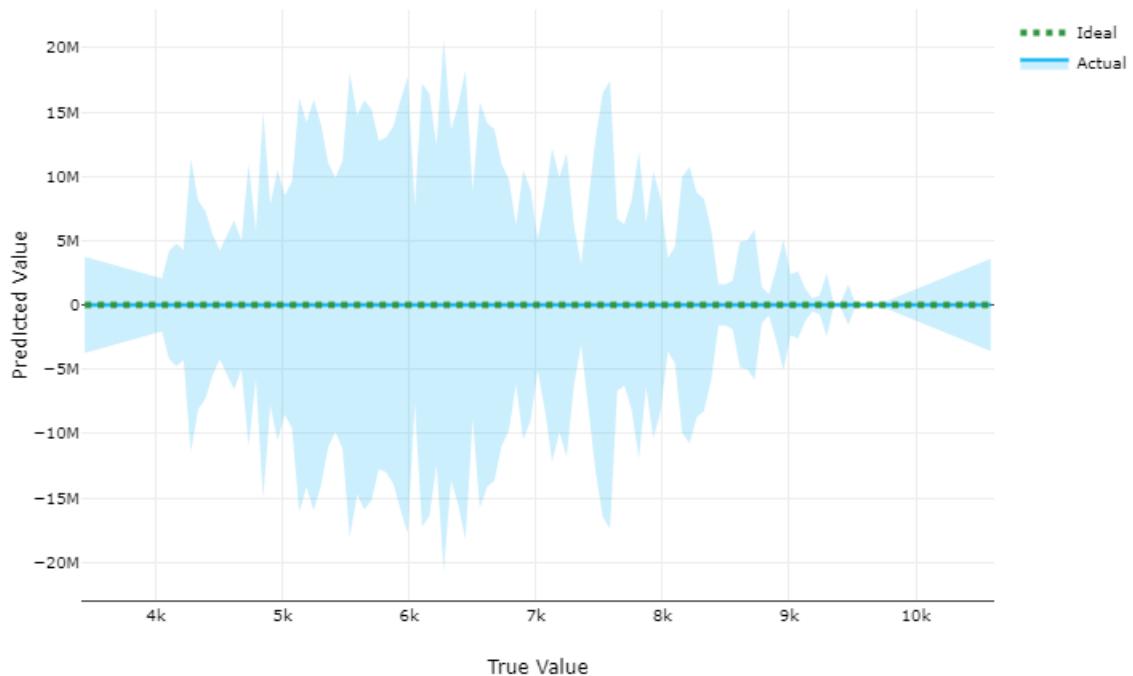
Example 1: A regression model with low accuracy in predictions

Predicted vs. True



Example 2: A regression model with high accuracy in its predictions

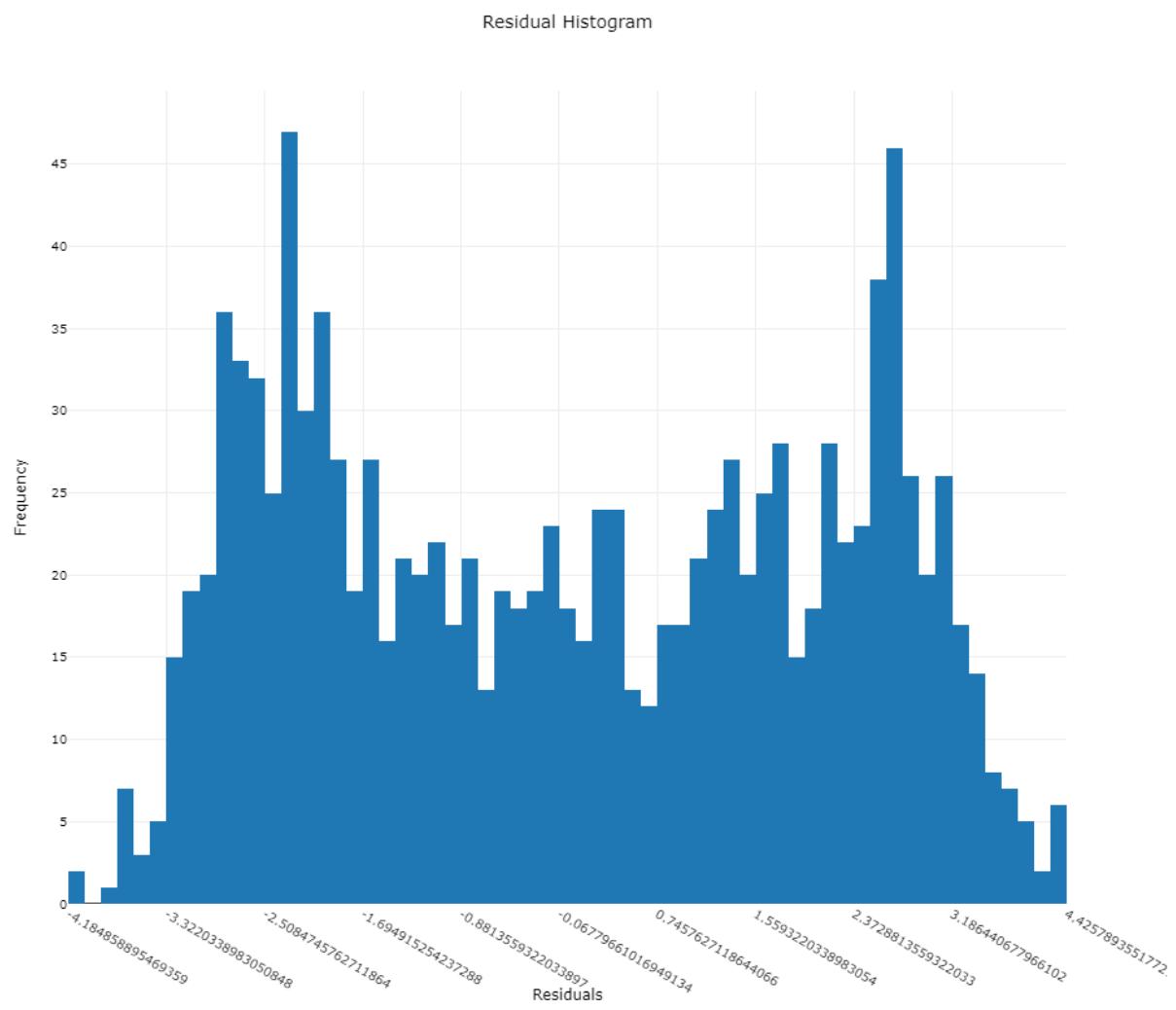
Predicted vs. True



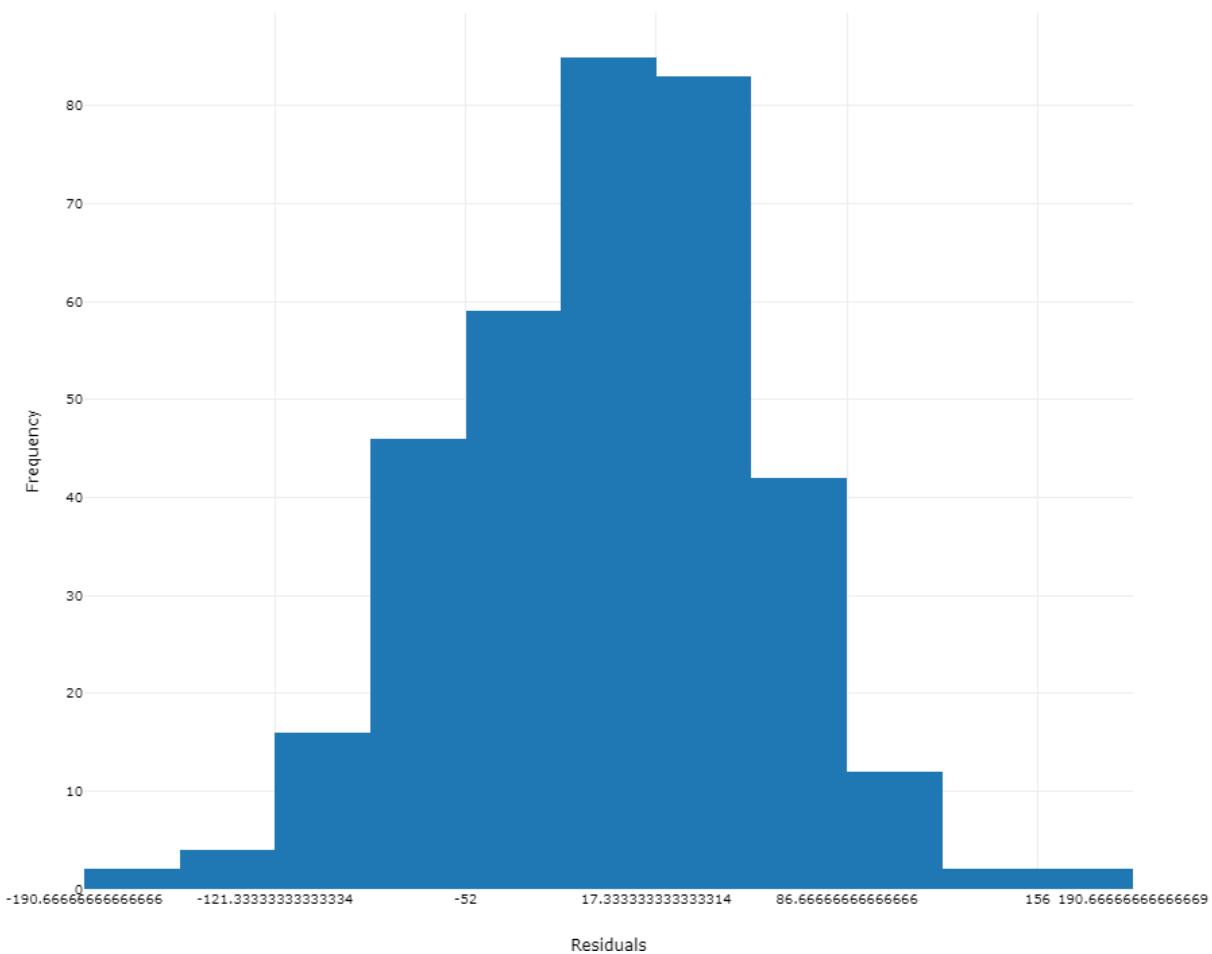
Histogram of residuals

A residual represents an observed y – the predicted \hat{y} . To show a margin of error with low bias, the histogram of residuals should be shaped as a bell curve, centered around 0.

Example 1: A regression model with bias in its errors

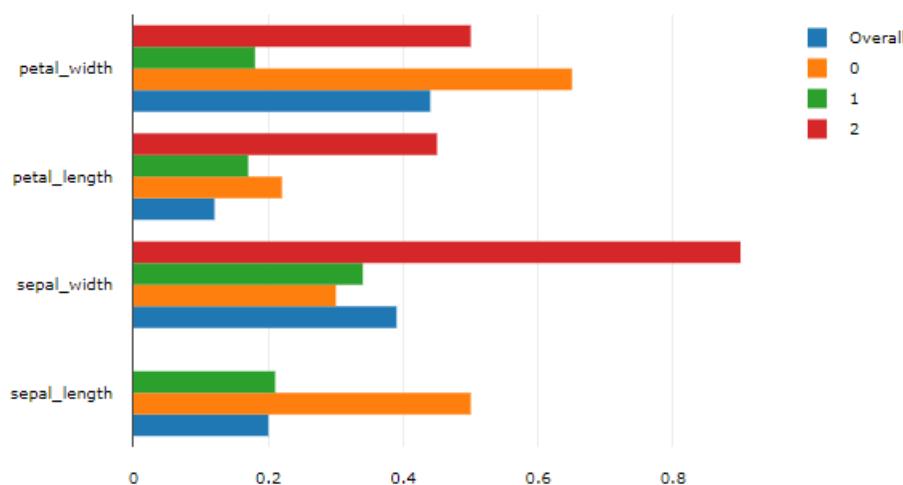


Example 2: A regression model with more even distribution of errors



Model explainability and feature importance

Feature importance gives a score that indicates how valuable each feature was in the construction of a model. You can review the feature importance score for the model overall as well as per class on a predictive model. You can see per feature how the importance compares against each class and overall.



Example notebooks

The following notebooks demonstrate concepts in this article:

- [how-to-use-azureml/training/train-within-notebook](#)
- [how-to-use-azureml/training/train-on-local](#)
- [how-to-use-azureml/training/logging-api/logging-api.ipynb](#)

Learn how to run notebooks by following the article, [Use Jupyter notebooks to explore this service](#).

Next steps

Try these next steps to learn how to use the Azure Machine Learning SDK for Python:

- See an example of how to register the best model and deploy it in the tutorial, [Train an image classification model with Azure Machine Learning](#).
- Learn how to [Train PyTorch Models with Azure Machine Learning](#).

Configure automated machine learning experiments

12/13/2018 • 18 minutes to read • [Edit Online](#)

Automated machine learning picks an algorithm and hyperparameters for you and generates a model ready for deployment. There are several options that you can use to configure automated machine learning experiments. In this guide, learn how to define various configuration settings.

To view examples of an automated machine learning experiments , see [Tutorial: Train a classification model with automated machine learning](#) or [Train models with automated machine learning in the cloud](#).

Configuration options available in automated machine learning:

- Select your experiment type: Classification, Regression or Forecasting
- Data source, formats, and fetch data
- Choose your compute target: local or remote
- Automated machine learning experiment settings
- Run an automated machine learning experiment
- Explore model metrics
- Register and deploy model

Select your experiment type

Before you begin your experiment, you should determine the kind of machine learning problem you are solving. Automated machine learning supports task types of classification, regression and forecasting.

While automated machine learning capabilities are generally available, **forecasting is still in public preview**.

Automated machine learning supports the following algorithms during the automation and tuning process. As a user, there is no need for you to specify the algorithm.

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

CLASSIFICATION	REGRESSION	FORECASTING
Gradient Boosting		
Light GBM		

Data source and format

Automated machine learning supports data that resides on your local desktop or in the cloud such as Azure Blob Storage. The data can be read into scikit-learn supported data formats. You can read the data into:

- Numpy arrays X (features) and y (target variable or also known as label)
- Pandas dataframe

Examples:

- Numpy arrays

```
digits = datasets.load_digits()
X_digits = digits.data
y_digits = digits.target
```

- Pandas dataframe

```
import pandas as pd
df = pd.read_csv("https://automldemos.blob.core.windows.net/datasets/PlayaEvents2016,_1.6MB,_3.4k-
rows.cleaned.2.tsv", delimiter="\t", quotechar='''')
# get integer labels
df = df.drop(["Label"], axis=1)
df_train, _, y_train, _ = train_test_split(df, y, test_size=0.1, random_state=42)
```

Fetch data for running experiment on remote compute

If you are using a remote compute to run your experiment, the data fetch must be wrapped in a separate python script `get_data()`. This script is run on the remote compute where the automated machine learning experiment is run. `get_data` eliminates the need to fetch the data over the wire for each iteration. Without `get_data`, your experiment will fail when you run on remote compute.

Here is an example of `get_data`:

```
%%writefile $project_folder/get_data.py
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder
def get_data(): # Burning man 2016 data
    df = pd.read_csv("https://automldemos.blob.core.windows.net/datasets/PlayaEvents2016,_1.6MB,_3.4k-
rows.cleaned.2.tsv", delimiter="\t", quotechar='''')
    # get integer labels
    le = LabelEncoder()
    le.fit(df["Label"].values)
    y = le.transform(df["Label"].values)
    df = df.drop(["Label"], axis=1)
    df_train, _, y_train, _ = train_test_split(df, y, test_size=0.1, random_state=42)
    return { "X" : df, "y" : y }
```

In your `AutoMLConfig` object, you specify the `data_script` parameter and provide the path to the `get_data` script

file similar to below:

```
automl_config = AutoMLConfig(***, data_script=project_folder + "/get_data.py", *** )
```

get_data script can return:

KEY	TYPE	MUTUALLY EXCLUSIVE WITH	DESCRIPTION
X	Pandas Dataframe or Numpy Array	data_train, label, columns	All features to train with
y	Pandas Dataframe or Numpy Array	label	Label data to train with. For classification, should be an array of integers.
X_valid	Pandas Dataframe or Numpy Array	data_train, label	<i>Optional</i> All features to validate with. If not specified, X is split between train and validate
y_valid	Pandas Dataframe or Numpy Array	data_train, label	<i>Optional</i> The label data to validate with. If not specified, y is split between train and validate
sample_weight	Pandas Dataframe or Numpy Array	data_train, label, columns	<i>Optional</i> A weight value for each sample. Use when you would like to assign different weights for your data points
sample_weight_valid	Pandas Dataframe or Numpy Array	data_train, label, columns	<i>Optional</i> A weight value for each validation sample. If not specified, sample_weight is split between train and validate
data_train	Pandas Dataframe	X, y, X_valid, y_valid	All data (features+label) to train with
label	string	X, y, X_valid, y_valid	Which column in data_train represents the label
columns	Array of strings		<i>Optional</i> Whitelist of columns to use for features
cv_splits_indices	Array of integers		<i>Optional</i> List of indexes to split the data for cross validation

Load and prepare data using DataPrep SDK

Automated machine learning experiments supports data loading and transforms using the dataprep SDK. Using the SDK provides the ability to

- 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

To learn about the data prep sdk refer the [How to prepare data for modeling article](#). Below is an example loading data using data prep sdk.

```
# The data referenced here was pulled from `sklearn.datasets.load_digits()`.  
simple_example_data_root = 'https://dprepdata.blob.core.windows.net/automl-notebook-data/'  
X = dprep.auto_read_file(simple_example_data_root + 'X.csv').skip(1) # Remove the header row.  
# You can use `auto_read_file` which intelligently figures out delimiters and datatypes of a file.  
  
# Here we read a comma delimited file and convert all columns to integers.  
y = dprep.read_csv(simple_example_data_root + 'y.csv').to_long(dprep.ColumnSelector(term='.*', use_regex = True))
```

Train and validation data

You can specify separate train and validation set either through `get_data()` or directly in the `AutoMLConfig` method.

Cross validation split options

K-Folds Cross Validation

Use `n_cross_validations` setting to specify the number of cross validations. The training data set will be randomly split into `n_cross_validations` folds of equal size. During each cross validation round, one of the folds will be used for validation of the model trained on the remaining folds. This process repeats for `n_cross_validations` rounds until each fold is used once as validation set. The average scores across all `n_cross_validations` rounds will be reported, and the corresponding model will be retrained on the whole training data set.

Monte Carlo Cross Validation (a.k.a. Repeated Random Sub-Sampling)

Use `validation_size` to specify the percentage of the training dataset that should be used for validation, and use `n_cross_validations` to specify the number of cross validations. During each cross validation round, a subset of size `validation_size` will be randomly selected for validation of the model trained on the remaining data. Finally, the average scores across all `n_cross_validations` rounds will be reported, and the corresponding model will be retrained on the whole training data set.

Custom validation dataset

Use custom validation dataset if random split is not acceptable (usually time series data or imbalanced data). You can specify your own validation dataset. The model will be evaluated against the validation dataset specified instead of random dataset.

Compute to run experiment

Next determine where the model will be trained. An automated machine learning training experiment can run on the following compute options:

- Your local machine such as a local desktop or laptop – Generally when you have small dataset and you are still in the exploration stage.
- A remote machine in the cloud – [Azure Machine Learning Managed Compute](#) is a managed service that enables the ability to train machine learning models on clusters of Azure virtual machines.

See the [Github site](#) for example notebooks with local and remote compute targets.

Configure your experiment settings

There are several options that you can use to configure your automated machine learning experiment. These parameters are set by instantiating an `AutoMLConfig` object.

Some examples include:

1. Classification experiment using AUC weighted as the primary metric with a max time of 12,000 seconds per iteration, with the experiment to end after 50 iterations and 2 cross validation folds.

```
automl_classifier = AutoMLConfig(  
    task='classification',  
    primary_metric='AUC_weighted',  
    max_time_sec=12000,  
    iterations=50,  
    X=X,  
    y=y,  
    n_cross_validations=2)
```

2. Below is an example of a regression experiment set to end after 100 iterations, with each iteration lasting up to 600 seconds with 5 validation cross folds.

```
automl_regressor = AutoMLConfig(  
    task='regression',  
    max_time_sec=600,  
    iterations=100,  
    primary_metric='r2_score',  
    X=X,  
    y=y,  
    n_cross_validations=5)
```

This table lists parameter settings available for your experiment and their default values.

PROPERTY	DESCRIPTION	DEFAULT VALUE
<code>task</code>	Specify the type of machine learning problem. Allowed values are <ul style="list-style-type: none">• Classification• Regression• Forecasting	None

PROPERTY	DESCRIPTION	DEFAULT VALUE
<code>primary_metric</code>	<p>Metric that you want to optimize in building your model. For example, if you specify accuracy as the <code>primary_metric</code>, automated machine learning looks to find a model with maximum accuracy. You can only specify one <code>primary_metric</code> per experiment. Allowed values are</p> <p>Classification:</p> <ul style="list-style-type: none"> • accuracy • AUC_weighted • precision_score_weighted • balanced_accuracy • average_precision_score_weighted <p>Regression:</p> <ul style="list-style-type: none"> • normalized_mean_absolute_error • spearman_correlation <p>normalized_root_mean_squared_error normalized_root_mean_squared_log_err or</p> <ul style="list-style-type: none"> • R2_score 	For Classification: accuracy For Regression: spearman_correlation
<code>experiment_exit_score</code>	You can set a target value for your <code>primary_metric</code> . Once a model is found that meets the <code>primary_metric</code> 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 <code>iterations</code> . Takes a double value. If the target never reaches, then Automated machine learning will continue until it reaches the number of iterations specified in <code>iterations</code> .	None
<code>iterations</code>	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
<code>max_concurrent_iterations</code>	Max number of iterations to run in parallel. This setting works only for remote compute.	1
<code>max_cores_per_iteration</code>	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

PROPERTY	DESCRIPTION	DEFAULT VALUE
<code>iteration_timeout_minutes</code>	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
<code>n_cross_validations</code>	Number of cross validation splits	None
<code>validation_size</code>	Size of validation set as percentage of all training sample.	None
<code>preprocess</code>	<p>True/False True enables experiment to perform preprocessing on the input. Following is a subset of preprocessing</p> <ul style="list-style-type: none"> • Missing 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 <p>Note : if data is sparse you cannot use preprocess = true</p>	False

PROPERTY	DESCRIPTION	DEFAULT VALUE
blacklist_models	<p>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.</p> <p>Allowed values for Classification</p> <ul style="list-style-type: none"> • LogisticRegression • SGD • MultinomialNaiveBayes • BernoulliNaiveBayes • SVM • LinearSVM • KNN • DecisionTree • RandomForest • ExtremeRandomTrees • LightGBM • GradientBoosting • TensorFlowDNN • TensorFlowLinearClassifier <p>Allowed values for Regression</p> <ul style="list-style-type: none"> • ElasticNet • GradientBoosting • DecisionTree • KNN • LassoLars • SGD • RandomForest • ExtremeRandomTree • LightGBM • TensorFlowLinearRegressor • TensorFlowDNN <p>Allowed values for Forecasting</p> <ul style="list-style-type: none"> • ElasticNet • GradientBoosting • DecisionTree • KNN • LassoLars • SGD • RandomForest • ExtremeRandomTree • LightGBM • TensorFlowLinearRegressor • TensorFlowDNN 	None

PROPERTY	DESCRIPTION	DEFAULT VALUE
<code>whitelist_models</code>	<p>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.</p> <p>Allowed values for Classification</p> <ul style="list-style-type: none"> • LogisticRegression • SGD • MultinomialNaiveBayes • BernoulliNaiveBayes • SVM • LinearSVM • KNN • DecisionTree • RandomForest • ExtremeRandomTrees • LightGBM • GradientBoosting • TensorFlowDNN • TensorFlowLinearClassifier <p>Allowed values for Regression</p> <ul style="list-style-type: none"> • ElasticNet • GradientBoosting • DecisionTree • KNN • LassoLars • SGD • RandomForest • ExtremeRandomTree • LightGBM • TensorFlowLinearRegressor • TensorFlowDNN <p>Allowed values for Forecasting</p> <ul style="list-style-type: none"> • ElasticNet • GradientBoosting • DecisionTree • KNN • LassoLars • SGD • RandomForest • ExtremeRandomTree • LightGBM • TensorFlowLinearRegressor • TensorFlowDNN 	None
<code>verbosity</code>	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.INFO

PROPERTY	DESCRIPTION	DEFAULT VALUE
<code>x</code>	All features to train with	None
<code>y</code>	Label data to train with. For classification, should be an array of integers.	None
<code>x_valid</code>	<i>Optional</i> All features to validate with. If not specified, X is split between train and validate	None
<code>y_valid</code>	<i>Optional</i> The label data to validate with. If not specified, y is split between train and validate	None
<code>sample_weight</code>	<i>Optional</i> A weight value for each sample. Use when you would like to assign different weights for your data points	None
<code>sample_weight_valid</code>	<i>Optional</i> A weight value for each validation sample. If not specified, sample_weight is split between train and validate	None
<code>run_configuration</code>	RunConfiguration object. Used for remote runs.	None
<code>data_script</code>	Path to a file containing the get_data method. Required for remote runs.	None
<code>model_explainability</code>	<i>Optional</i> 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
<code>enable_ensembling</code>	Flag to enable an ensembling iteration after all the other iterations complete.	True
<code>ensemble_iterations</code>	Number of iterations during which we choose a fitted pipeline to be part of the final ensemble.	15
<code>experiment_timeout_minutes</code>	Limits the amount of time (minutes) that the whole experiment run can take	None

Data pre-processing and featurization

If you use `preprocess=True`, the following data preprocessing steps are performed automatically for you:

1. Drop high cardinality or no variance features

- Drop features with no useful information from training and validation sets. These include features with all values missing, same value across all rows or with extremely high cardinality (e.g., hashes, IDs or

GUIDs).

2. Missing value imputation

- For numerical features, impute missing values with average of values in the column.
- For categorical features, impute missing values with most frequent value.

3. Generate additional features

- For DateTime features: Year, Month, Day, Day of week, Day of year, Quarter, Week of the year, Hour, Minute, Second.
- For Text features: Term frequency based on word unigram, bi-grams, and tri-gram, Count vectorizer.

4. Transformations and encodings

- Numeric features with very few unique values transformed into categorical features.
- Depending on cardinality of categorical features, perform label encoding or (hashing) one-hot encoding.

Run experiment

Submit the experiment to run and generate a model. Pass the `AutoMLConfig` to the `submit` method to generate the model.

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

NOTE

Dependencies are first installed on a new machine. It may take up to 10 minutes before output is shown. Setting `show_output` to `True` results in output being shown on the console.

Explore model metrics

You can view your results in a widget or inline if you are in a notebook. See [Track and evaluate models](#) for more details.

Classification metrics

The following metrics are saved in each iteration for a classification task.

PRIMARY METRIC	DESCRIPTION	CALCULATION	EXTRA PARAMETERS
AUC_Macro	AUC is the Area under the Receiver Operating Characteristic Curve. Macro is the arithmetic mean of the AUC for each class.	Calculation	average="macro"
AUC_Micro	AUC is the Area under the Receiver Operating Characteristic Curve. Micro is computed globally by combining the true positives and false positives from each class	Calculation	average="micro"

PRIMARY METRIC	DESCRIPTION	CALCULATION	EXTRA PARAMETERS
AUC_Weighted	AUC is the Area under the Receiver Operating Characteristic Curve. Weighted is the arithmetic mean of the score for each class, weighted by the number of true instances in each class	Calculation	average="weighted"
accuracy	Accuracy is the percent of predicted labels that exactly match the true labels.	Calculation	None
average_precision_score_macro	Average precision summarizes a precision-recall curve as the weighted mean of precisions achieved at each threshold, with the increase in recall from the previous threshold used as the weight. Macro is the arithmetic mean of the average precision score of each class	Calculation	average="macro"
average_precision_score_micro	Average precision summarizes a precision-recall curve as the weighted mean of precisions achieved at each threshold, with the increase in recall from the previous threshold used as the weight. Micro is computed globally by combining the true positives and false positives at each cutoff	Calculation	average="micro"
average_precision_score_weighted	Average precision summarizes a precision-recall curve as the weighted mean of precisions achieved at each threshold, with the increase in recall from the previous threshold used as the weight. Weighted is the arithmetic mean of the average precision score for each class, weighted by the number of true instances in each class	Calculation	average="weighted"
balanced_accuracy	Balanced accuracy is the arithmetic mean of recall for each class.	Calculation	average="macro"

PRIMARY METRIC	DESCRIPTION	CALCULATION	EXTRA PARAMETERS
f1_score_macro	F1 score is the harmonic mean of precision and recall. Macro is the arithmetic mean of F1 score for each class	Calculation	average="macro"
f1_score_micro	F1 score is the harmonic mean of precision and recall. Micro is computed globally by counting the total true positives, false negatives, and false positives	Calculation	average="micro"
f1_score_weighted	F1 score is the harmonic mean of precision and recall. Weighted mean by class frequency of F1 score for each class	Calculation	average="weighted"
log_loss	This is the loss function used in (multinomial) logistic regression and extensions of it such as neural networks, defined as the negative log-likelihood of the true labels given a probabilistic classifier's predictions. For a single sample with true label y_t in {0,1} and estimated probability y_p that $y_t = 1$, the log loss is $-\log P(y_t y_p) = -(y_t \log(y_p) + (1 - y_t) \log(1 - y_p))$	Calculation	None
norm_macro_recall	Normalized Macro Recall is Macro Recall normalized so that random performance has a score of 0 and perfect performance has a score of 1. This is achieved by <pre>norm_macro_recall := (recall_score_macro - R)/(1 - R)</pre> , where R is the expected value of recall_score_macro for random predictions (i.e., R=0.5 for binary classification and R=(1/C) for C-class classification problems)	Calculation	average = "macro" and then $(\text{recall_score_macro} - R)/(1 - R)$, where R is the expected value of recall_score_macro for random predictions (i.e., R=0.5 for binary classification and R=(1/C) for C-class classification problems)
precision_score_macro	Precision is the percent of elements labeled as a certain class that actually are in that class. Macro is the arithmetic mean of precision for each class	Calculation	average="macro"

PRIMARY METRIC	DESCRIPTION	CALCULATION	EXTRA PARAMETERS
precision_score_micro	Precision is the percent of elements labeled as a certain class that actually are in that class. Micro is computed globally by counting the total true positives and false positives	Calculation	average="micro"
precision_score_weighted	Precision is the percent of elements labeled as a certain class that actually are in that class. Weighted is the arithmetic mean of precision for each class, weighted by number of true instances in each class	Calculation	average="weighted"
recall_score_macro	Recall is the percent of elements actually in a certain class that are correctly labeled. Macro is the arithmetic mean of recall for each class	Calculation	average="macro"
recall_score_micro	Recall is the percent of elements actually in a certain class that are correctly labeled. Micro is computed globally by counting the total true positives, false negatives	Calculation	average="micro"
recall_score_weighted	Recall is the percent of elements actually in a certain class that are correctly labeled. Weighted is the arithmetic mean of recall for each class, weighted by number of true instances in each class	Calculation	average="weighted"
weighted_accuracy	Weighted accuracy is accuracy where the weight given to each example is equal to the proportion of true instances in that example's true class	Calculation	sample_weight is a vector equal to the proportion of that class for each element in the target

Regression and forecasting metrics

The following metrics are saved in each iteration for a regression or forecasting task.

PRIMARY METRIC	DESCRIPTION	CALCULATION	EXTRA PARAMETERS
----------------	-------------	-------------	------------------

PRIMARY METRIC	DESCRIPTION	CALCULATION	EXTRA PARAMETERS
explained_variance	Explained variance is the proportion to which a mathematical model accounts for the variation of a given data set. It is the percent decrease in variance of the original data to the variance of the errors. When the mean of the errors is 0, it is equal to explained variance.	Calculation	None
r2_score	R2 is the coefficient of determination or the percent reduction in squared errors compared to a baseline model that outputs the mean. When the mean of the errors is 0, it is equal to explained variance.	Calculation	None
spearman_correlation	Spearman correlation is a nonparametric measure of the monotonicity of the relationship between two datasets. Unlike the Pearson correlation, the Spearman correlation does not assume that both datasets are normally distributed. Like other correlation coefficients, this one varies between -1 and +1 with 0 implying no correlation. Correlations of -1 or +1 imply an exact monotonic relationship. Positive correlations imply that as x increases, so does y. Negative correlations imply that as x increases, y decreases.	Calculation	None
mean_absolute_error	Mean absolute error is the expected value of absolute value of difference between the target and the prediction	Calculation	None
normalized_mean_absolute_error	Normalized mean absolute error is mean Absolute Error divided by the range of the data	Calculation	Divide by range of the data
median_absolute_error	Median absolute error is the median of all absolute differences between the target and the prediction. This loss is robust to outliers.	Calculation	None

Primary Metric	Description	Calculation	Extra Parameters
normalized_median_absolute_error	Normalized median absolute error is median absolute error divided by the range of the data	Calculation	Divide by range of the data
root_mean_squared_error	Root mean squared error is the square root of the expected squared difference between the target and the prediction	Calculation	None
normalized_root_mean_squared_error	Normalized root mean squared error is root mean squared error divided by the range of the data	Calculation	Divide by range of the data
root_mean_squared_log_err or	Root mean squared log error is the square root of the expected squared logarithmic error	Calculation	None
normalized_root_mean_squared_log_error	Normalized Root mean squared log error is root mean squared log error divided by the range of the data	Calculation	Divide by range of the data

Explain the model

While automated machine learning capabilities are generally available, **the model explainability feature is still in public preview.**

Automated machine learning allows you to understand feature importance. During the training process, you can get global feature importance for the model. For classification scenarios, you can also get class-level feature importance. You must provide a validation dataset (X_valid) to get feature importance.

There are two ways to generate feature importance.

- Once an experiment is complete, you can use `explain_model` method on any iteration.

```
from azureml.train.automl.automlexplainer import explain_model

shap_values, expected_values, overall_summary, overall_imp, per_class_summary, per_class_imp = \
    explain_model(fitted_model, X_train, X_test)

#Overall feature importance
print(overall_imp)
print(overall_summary)

#Class-level feature importance
print(per_class_imp)
print(per_class_summary)
```

- To view feature importance for all iterations, set `model_explainability` flag to `True` in AutoMLConfig.

```
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)
```

Once done, you can use retrieve_model_explanation method to retrieve feature importance for a specific iteration.

```
from azureml.train.automl.automlexplainer import retrieve_model_explanation

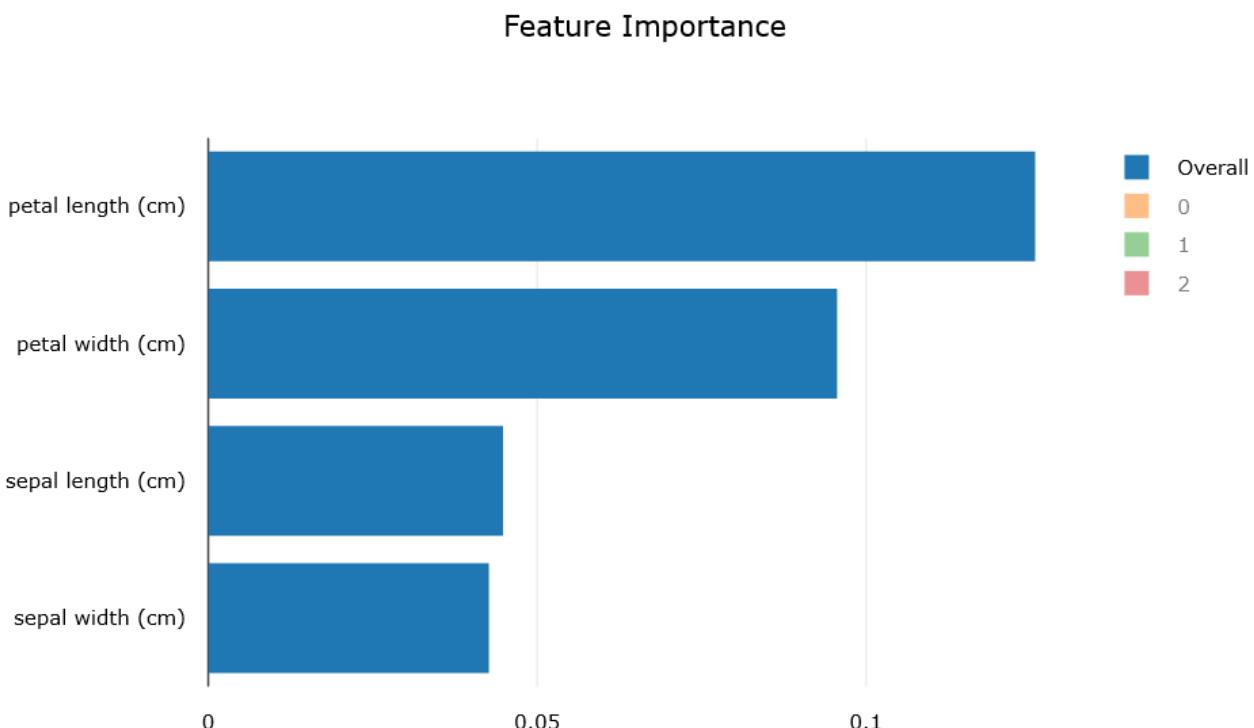
shap_values, expected_values, overall_summary, overall_imp, per_class_summary, per_class_imp = \
    retrieve_model_explanation(best_run)

#Overall feature importance
print(overall_imp)
print(overall_summary)

#Class-level feature importance
print(per_class_imp)
print(per_class_summary)
```

You can visualize the feature importance chart in your workspace in the Azure portal. The chart is also shown when using the Jupyter widget in a notebook. To learn more about the charts refer to the [Sample Azure ML notebooks article](#).

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



Next steps

Learn more about [how and where to deploy a model](#).

Learn more about [how to train a classification model with Automated machine learning](#) or [how to train using Automated machine learning on a remote resource](#).

Train models with automated machine learning in the cloud

12/11/2018 • 7 minutes to read • [Edit Online](#)

In Azure Machine Learning, you train your model on different types of compute resources that you manage. The compute target could be a local computer or a computer in the cloud.

You can easily scale up or scale out your machine learning experiment by adding additional compute targets. Compute target options include Ubuntu-based Data Science Virtual Machine (DSVM) or Azure Machine Learning Compute. The DSVM is a customized VM image on Microsoft's Azure cloud built specifically for doing data science. It has many popular data science and other tools pre-installed and pre-configured.

In this article, you learn how to build a model using automated ML on the DSVM.

How does remote differ from local?

The tutorial "[Train a classification model with automated machine learning](#)" teaches you how to use a local computer to train model with automated ML. The workflow when training locally also applies to remote targets as well. However, with remote compute, automated ML experiment iterations are executed asynchronously. This functionality allows you to cancel a particular iteration, watch the status of the execution, or continue to work on other cells in the Jupyter notebook. To train remotely, you first create a remote compute target such as an Azure DSVM. Then you configure the remote resource and submit your code there.

This article shows the extra steps needed to run an automated ML experiment on a remote DSVM. The workspace object, `ws`, from the tutorial is used throughout the code here.

```
ws = Workspace.from_config()
```

Create resource

Create the DSVM in your workspace (`ws`) if it doesn't already exist. If the DSVM was previously created, this code skips the creation process and loads the existing resource detail into the `dsvm_compute` object.

Time estimate: Creation of the VM takes approximately 5 minutes.

```
from azureml.core.compute import DsvmCompute

dsvm_name = 'mydsvm' #Name your DSVM
try:
    dsvm_compute = DsvmCompute(ws, dsvm_name)
    print('found existing dsvm.')
except:
    print('creating new dsvm.')
    # Below is using a VM of SKU Standard_D2_v2 which is 2 core machine. You can check Azure virtual machines documentation for additional SKUs of VMs.
    dsvm_config = DsvmCompute.provisioning_configuration(vm_size = "Standard_D2_v2")
    dsvm_compute = DsvmCompute.create(ws, name = dsvm_name, provisioning_configuration = dsvm_config)
    dsvm_compute.wait_for_completion(show_output = True)
```

You can now use the `dsvm_compute` object as the remote compute target.

DSVM name restrictions include:

- Must be shorter than 64 characters.
- Cannot include any of the following characters: \ ~ ! @ # \$ % ^ & * () = + _ [] { } \\ | ; ' " , < > / ? `

WARNING

If creation fails with a message about Marketplace purchase eligibility:

1. Go to the [Azure portal](#)
2. Start creating a DSVM
3. Select "Want to create programmatically" to enable programmatic creation
4. Exit without actually creating the VM
5. Rerun the creation code

This code doesn't create a user name or password for the DSVM that is provisioned. If you want to connect directly to the VM, go to the [Azure portal](#) to create credentials.

Attach existing Linux DSVM

You can also attach an existing Linux DSVM as the compute target. This example utilizes an existing DSVM, but doesn't create a new resource.

NOTE

The following code uses the `RemoteCompute` target class to attach an existing VM as your compute target. The `DsvmCompute` class will be deprecated in future releases in favor of this design pattern.

Run the following code to create the compute target from a pre-existing Linux DSVM.

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

attach_config = RemoteCompute.attach_configuration(username='<username>',
                                                 address='<ip_address_or_fqdn>',
                                                 ssh_port=22,
                                                 private_key_file='./ssh/id_rsa')

compute_target = ComputeTarget.attach(workspace=ws,
                                      name='attached_vm',
                                      attach_configuration=attach_config)

compute_target.wait_for_completion(show_output=True)
```

You can now use the `compute_target` object as the remote compute target.

Access data using get_data file

Provide the remote resource access to your training data. For automated machine learning experiments running on remote compute, the data needs to be fetched using a `get_data()` function.

To provide access, you must:

- Create a `get_data.py` file containing a `get_data()` function
- Place that file in a directory accessible as an absolute path

You can encapsulate code to read data from a blob storage or local disk in the `get_data.py` file. In the following code sample, the data comes from the `sklearn` package.

WARNING

If you are using remote compute, then you must use `get_data()` where your data transformations are performed. If you need to install additional libraries for data transformations as part of `get_data()`, there are additional steps to be followed. Refer to the [auto-ml-dataprep sample notebook](#) for details.

```
# Create a project_folder if it doesn't exist
if not os.path.exists(project_folder):
    os.makedirs(project_folder)

#Write the get_data file.
%%writefile $project_folder/get_data.py

from sklearn import datasets
from scipy import sparse
import numpy as np

def get_data():

    digits = datasets.load_digits()
    X_digits = digits.data[10:,:]
    y_digits = digits.target[10:]

    return { "X" : X_digits, "y" : y_digits }
```

Configure experiment

Specify the settings for `AutoMLConfig`. (See a [full list of parameters](#) and their possible values.)

In the settings, `run_configuration` is set to the `run_config` object, which contains the settings and configuration for the DSVM.

```
from azureml.train.automl import AutoMLConfig
import time
import logging

automl_settings = {
    "name": "AutoML_Demo_Experiment_{0}".format(time.time()),
    "iteration_timeout_minutes": 10,
    "iterations": 20,
    "n_cross_validations": 5,
    "primary_metric": 'AUC_weighted',
    "preprocess": False,
    "max_concurrent_iterations": 10,
    "verbosity": logging.INFO
}

automl_config = AutoMLConfig(task='classification',
                             debug_log='automl_errors.log',
                             path=project_folder,
                             compute_target = dsvm_compute,
                             data_script=project_folder + "/get_data.py",
                             **automl_settings,
                             )
```

Enable model explanations

Set the optional `model_explainability` parameter in the `AutoMLConfig` constructor. Additionally, a validation dataframe object must be passed as a parameter `X_valid` to use the model explainability feature.

```

automl_config = AutoMLConfig(task='classification',
                             debug_log='automl_errors.log',
                             path=project_folder,
                             compute_target = dsvm_compute,
                             data_script=project_folder + "/get_data.py",
                             **automl_settings,
                             model_explainability=True,
                             X_valid = X_test
)

```

Submit training experiment

Now submit the configuration to automatically select the algorithm, hyper parameters, and train the model.

```

from azureml.core.experiment import Experiment
experiment=Experiment(ws, 'automl_remote')
remote_run = experiment.submit(automl_config, show_output=True)

```

You will see output similar to the following example:

```

Running on remote compute: mydsvmParent Run ID: AutoML_015ffe76-c331-406d-9bfd-0fd42d8ab7f6
*****
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
2              Standardize SGD classifier 0:02:36    0.954       0.954
7              Normalizer DT           0:02:22    0.161       0.954
0              Scale MaxAbs 1 extra trees 0:02:45    0.936       0.954
4              Robust Scaler SGD classifier 0:02:24    0.867       0.954
1              Normalizer kNN          0:02:44    0.984       0.984
9              Normalizer extra trees 0:03:15    0.834       0.984
5              Robust Scaler DT          0:02:18    0.736       0.984
8              Standardize kNN          0:02:05    0.981       0.984
6              Standardize SVM          0:02:18    0.984       0.984
10             Scale MaxAbs 1 DT          0:02:18    0.077       0.984
11             Standardize SGD classifier 0:02:24    0.863       0.984
3              Standardize gradient boosting 0:03:03    0.971       0.984
12             Robust Scaler logistic regression 0:02:32    0.955       0.984
14             Scale MaxAbs 1 SVM          0:02:15    0.989       0.989
13             Scale MaxAbs 1 gradient boosting 0:02:15    0.971       0.989
15             Robust Scaler kNN          0:02:28    0.904       0.989
17             Standardize kNN          0:02:22    0.974       0.989
16             Scale 0/1 gradient boosting 0:02:18    0.968       0.989
18             Scale 0/1 extra trees        0:02:18    0.828       0.989
19             Robust Scaler kNN          0:02:32    0.983       0.989

```

Explore results

You can use the same Jupyter widget as the one in [the training tutorial](#) to see a graph and table of results.

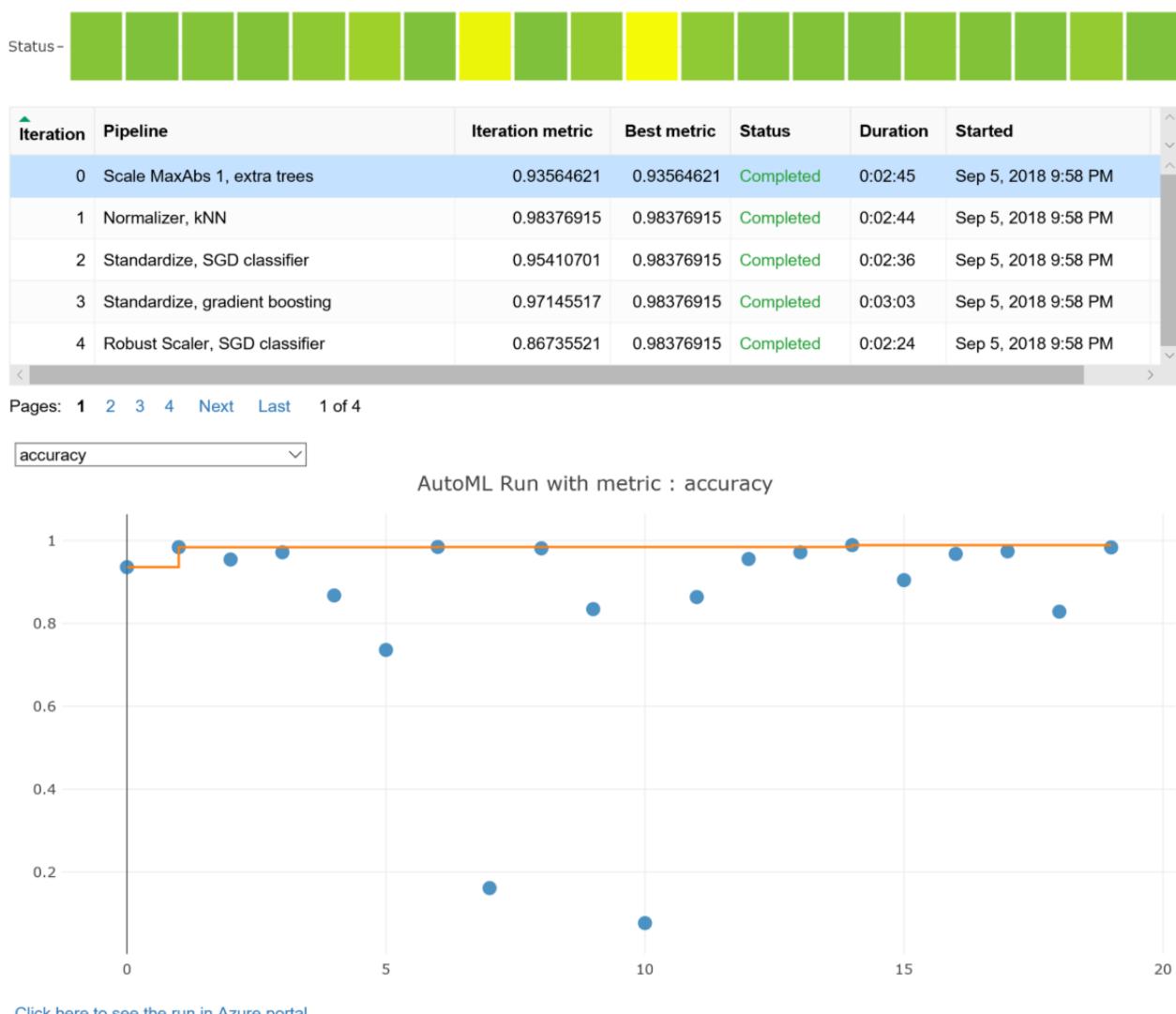
```

from azureml.widgets import RunDetails
RunDetails(remote_run).show()

```

Here is a static image of the widget. In the notebook, you can click on any line in the table to see run properties

and output logs for that run. You can also use the dropdown above the graph to view a graph of each available metric for each iteration.



The widget displays a URL you can use to see and explore the individual run details.

View logs

Find logs on the DSVM under `/tmp/azureml_run/{iterationid}/azureml-logs`.

Best model explanation

Retrieving model explanation data allows you to see detailed information about the models to increase transparency into what's running on the back-end. In this example, you run model explanations only for the best fit model. If you run for all models in the pipeline, it will result in significant run time. Model explanation information includes:

- `shap_values`: The explanation information generated by shap lib
- `expected_values`: The expected value of the model applied to set of `X_train` data.
- `overall_summary`: The model level feature importance values sorted in descending order
- `overall_imp`: The feature names sorted in the same order as in `overall_summary`
- `per_class_summary`: The class level feature importance values sorted in descending order. Only available for the classification case
- `per_class_imp`: The feature names sorted in the same order as in `per_class_summary`. Only available for the classification case

Use the following code to select the best pipeline from your iterations. The `get_output` method returns the best

run and the fitted model for the last fit invocation.

```
best_run, fitted_model = remote_run.get_output()
```

Import the `retrieve_model_explanation` function and run on the best model.

```
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)
```

Print results for the `best_run` explanation variables you want to view.

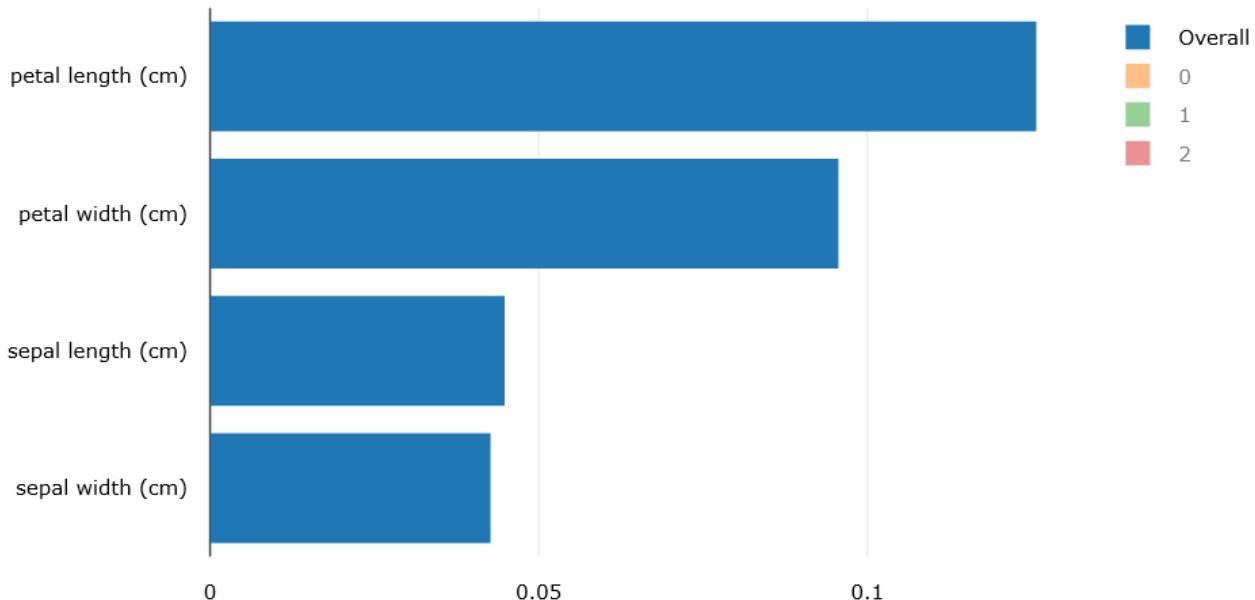
```
print(overall_summary)
print(overall_imp)
print(per_class_summary)
print(per_class_imp)
```

Printing the `best_run` explanation summary variables results in the following output.

```
[0.13751896361164054, 0.12196113064848417, 0.02515800367003573, 0.0204136524293061]
['petal width (cm)', 'petal length (cm)', 'sepal length (cm)', 'sepal width (cm)']
[[0.13935022276497785, 0.12224503499384598, 0.030073645391740296, 0.028297085859246965], [0.14292484561842992, 0.11960261236451923, 0.02612530890966221, 0.02358220878544405], [0.14738701023064565, 0.10693055681595545, 0.02181815683292284, 0.00681856251900913]]
[['petal length (cm)', 'petal width (cm)', 'sepal length (cm)', 'sepal width (cm)'], ['petal width (cm)', 'petal length (cm)', 'sepal width (cm)', 'sepal length (cm)'], ['petal width (cm)', 'petal length (cm)', 'sepal length (cm)', 'sepal width (cm)']]
```

You can also visualize feature importance through the widget UI as well as the web UI on Azure portal inside your workspace.

Feature Importance



Example

The [how-to-use-azureml/automated-machine-learning/remote-execution/auto-ml-remote-execution.ipynb](#) notebook demonstrates concepts in this article.

Learn how to run notebooks by following the article, [Use Jupyter notebooks to explore this service](#).

Next steps

Learn [how to configure settings for automatic training](#).

ONNX and Azure Machine Learning: Create and deploy interoperable AI models

12/10/2018 • 4 minutes to read • [Edit Online](#)

The [Open Neural Network Exchange \(ONNX\)](#) format is an open standard for representing machine learning models. ONNX is supported by a [community of partners](#), including Microsoft, who create compatible frameworks and tools. Microsoft is committed to open and interoperable AI so that data scientists and developers can:

- Use the framework of their choice to create and train models
- Deploy models cross-platform with minimal integration work

Microsoft supports ONNX across its products including Azure and Windows to help you achieve these goals.

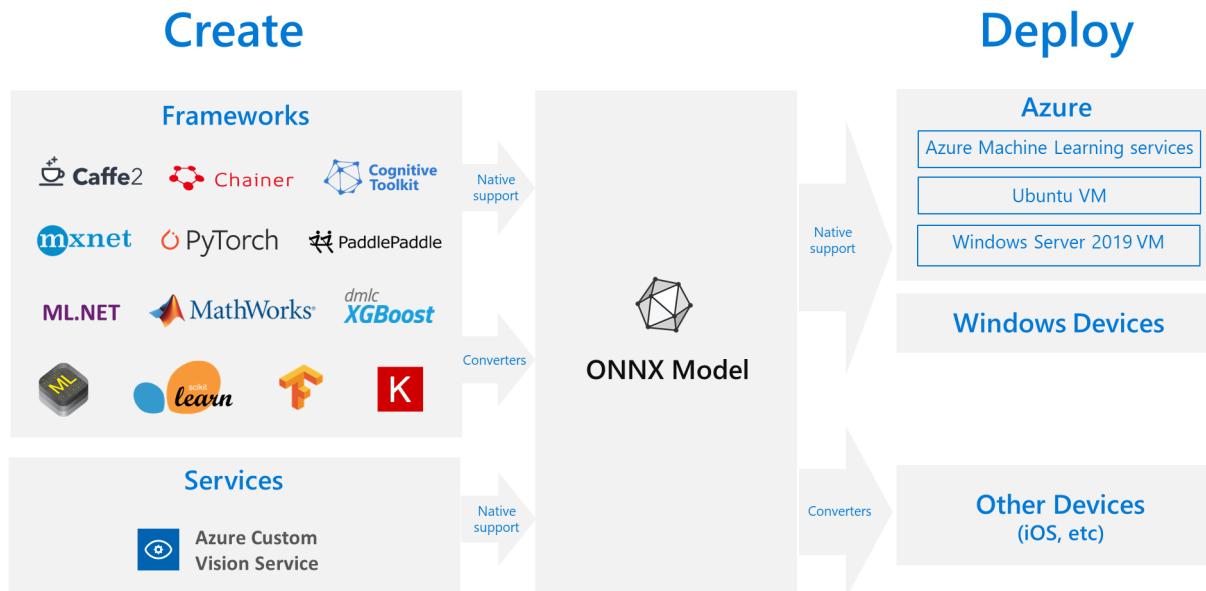
Why choose ONNX?

The interoperability you get with ONNX makes it possible to get great ideas into production faster. With ONNX, data scientists can choose their preferred framework for the job. Similarly, developers can spend less time getting models ready for production, and deploy across the cloud and edge.

You can create ONNX models from many frameworks, including PyTorch, Chainer, Microsoft Cognitive Toolkit (CNTK), MXNet, ML.NET, TensorFlow, Keras, SciKit-Learn, and more.

There is also an ecosystem of tools for visualizing and accelerating ONNX models. A number of pre-trained ONNX models are also available for common scenarios.

ONNX models can be deployed to the cloud using Azure Machine Learning and ONNX Runtime. They can also be deployed to Windows 10 devices using [Windows ML](#). They can even be deployed to other platforms using converters that are available from the ONNX community.



Get ONNX models

You can obtain ONNX models in several ways:

- Get a pre-trained ONNX model from the [ONNX Model Zoo](#) (see example at the bottom of this article)

- Generate a customized ONNX model from [Azure Custom Vision service](#)
- Convert existing model from another format to ONNX (see example at the bottom of this article)
- Train a new ONNX model in Azure Machine Learning service (see example at the bottom of this article)

Save/convert your models to ONNX

You can convert existing models to ONNX or save them as ONNX at the end of your training.

FRAMEWORK FOR MODEL	CONVERSION EXAMPLE OR TOOL
PyTorch	Jupyter notebook
Microsoft Cognitive Toolkit (CNTK)	Jupyter notebook
TensorFlow	tensorflow-onnx converter
Chainer	Jupyter notebook
MXNet	Jupyter notebook
Keras, ScitKit-Learn, CoreML XGBoost, and libSVM	WinMLTools

You can find the latest list of supported frameworks and converters at the [ONNX Tutorials site](#).

Deploy ONNX models in Azure

With Azure Machine Learning service, you can deploy, manage, and monitor your ONNX models. Using the standard [deployment workflow](#) and ONNX Runtime, you can create a REST endpoint hosted in the cloud. See a full example Jupyter notebook at the end of this article to try it out for yourself.

Install and configure ONNX Runtime

ONNX Runtime is an open source high-performance inference engine for ONNX models. It provides hardware acceleration on both CPU and GPU, with APIs available for Python, C#, and C. ONNX Runtime supports ONNX 1.2+ models and runs on Linux, Windows, and Mac. Python packages are available on [PyPi.org](#) ([CPU](#), [GPU](#)), and [C# package](#) is on [Nuget.org](#). See more about the project on [Github](#).

To install ONNX Runtime for Python, use:

```
pip install onnxruntime
```

To call ONNX Runtime in your Python script, use:

```
import onnxruntime
session = onnxruntime.InferenceSession("path to model")
```

The documentation accompanying the model usually tells you the inputs and outputs for using the model. You can also use a visualization tool such as [Netron](#) to view the model. ONNX Runtime also lets you query the model metadata, inputs, and outputs:

```
session.get_modelmeta()
first_input_name = session.get_inputs()[0].name
first_output_name = session.get_outputs()[0].name
```

To inference your model, use `run` and pass in the list of outputs you want returned (leave empty if you want all of them) and a map of the input values. The result is a list of the outputs.

```
results = session.run(["output1", "output2"], {"input1": indata1, "input2": indata2})
results = session.run([], {"input1": indata1, "input2": indata2})
```

For the complete Python API reference, see the [ONNX Runtime reference docs](#).

Example deployment steps

Here is an example for deploying an ONNX model:

1. Initialize your Azure Machine Learning service workspace. If you don't have one yet, learn how to create a workspace in [this quickstart](#).

```
from azureml.core import Workspace

ws = Workspace.from_config()
print(ws.name, ws.resource_group, ws.location, ws.subscription_id, sep = '\n')
```

2. Register the model with Azure Machine Learning.

```
from azureml.core.model import Model

model = Model.register(model_path = "model.onnx",
                      model_name = "MyONNXmodel",
                      tags = ["onnx"],
                      description = "test",
                      workspace = ws)
```

3. Create an image with the model and any dependencies.

```
from azureml.core.image import ContainerImage

image_config = ContainerImage.image_configuration(execution_script = "score.py",
                                                 runtime = "python",
                                                 conda_file = "myenv.yml",
                                                 description = "test",
                                                 tags = ["onnx"]
)

image = ContainerImage.create(name = "myonnxmodelimage",
                             # this is the model object
                             models = [model],
                             image_config = image_config,
                             workspace = ws)

image.wait_for_creation(show_output = True)
```

The file `score.py` contains the scoring logic and needs to be included in the image. This file is used to run the model in the image. See this [tutorial](#) for instructions on how to create a scoring script. An example file for an ONNX model is shown below:

```

import onnxruntime
import json
import numpy as np
import sys
from azureml.core.model import Model

def init():
    global model_path
    model_path = Model.get_model_path(model_name = 'MyONNXmodel')

def run(raw_data):
    try:
        data = json.loads(raw_data)['data']
        data = np.array(data)

        sess = onnxruntime.InferenceSession(model_path)
        result = sess.run(["outY"], {"inX": data})

        return json.dumps({"result": result.tolist()})
    except Exception as e:
        result = str(e)
        return json.dumps({"error": result})

```

The file `myenv.yml` describes the dependencies needed for the image. See this [tutorial](#) for instructions on how to create an environment file, such as this sample file:

```

from azureml.core.conda_dependencies import CondaDependencies

myenv = CondaDependencies()
myenv.add_pip_package("numpy")
myenv.add_pip_package("azureml-core")
myenv.add_pip_package("onnxruntime")

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

```

4. To deploy your model, see the [How to deploy and where](#) document.

Examples

See [how-to-use-azureml/deployment/onnx](#) for example notebooks that create and deploy ONNX models.

Learn how to run notebooks by following the article, [Use Jupyter notebooks to explore this service](#).

More info

Learn more about ONNX or contribute to the project:

- [ONNX project website](#)
- [ONNX code on GitHub](#)

Learn more about ONNX Runtime or contribute to the project:

- [ONNX Runtime Github Repo](#)

Deploy models with the Azure Machine Learning service

12/13/2018 • 12 minutes to read • [Edit Online](#)

The Azure Machine Learning service provides several ways you can deploy your trained model using the SDK. In this document, learn how to deploy your model as a web service in the Azure cloud, or to IoT edge devices.

You can deploy models to the following compute targets:

COMPUTE TARGET	DEPLOYMENT TYPE	DESCRIPTION
Azure Container Instances (ACI)	Web service	Fast deployment. Good for development or testing.
Azure Kubernetes Service (AKS)	Web service	Good for high-scale production deployments. Provides autoscaling, and fast response times.
Azure IoT Edge	IoT module	Deploy models on IoT devices. Inferencing happens on the device.
Field-programmable gate array (FPGA)	Web service	Ultra-low latency for real-time inferencing.

Prerequisites

- An Azure Machine Learning service workspace and the Azure Machine Learning SDK for Python installed. Learn how to get these prerequisites using the [Get started with Azure Machine Learning quickstart](#).
- A trained model in either pickle (`.pk1`) or ONNX (`.onnx`) format. If you do not have a trained model, use the steps in the [Train models](#) tutorial to train and register one with the Azure Machine Learning service.
- The code sections assume that `ws` references your machine learning workspace. For example,
`ws = Workspace.from_config()`.

Deployment workflow

The process of deploying a model is similar for all compute targets:

1. Train a model.
2. Register the model.
3. Create an image configuration.
4. Create the image.
5. Deploy the image to a compute target.
6. Test the deployment

7. (Optional) Clean up artifacts.

- When **deploying as a web service**, there are three deployment options:
 - deploy**: When using this method, you do not need to register the model or create the image. However you cannot control the name of the model or image, or associated tags and descriptions.
 - deploy_from_model**: When using this method, you do not need to create an image. But you do not have control over the name of the image that is created.
 - deploy_from_image**: Register the model and create an image before using this method.

The examples in this document use `deploy_from_image`.

- When **deploying as an IoT Edge module**, you must register the model and create the image.

Register a model

Only trained models can be deployed. The model can be trained using Azure Machine Learning, or another service. To register a model from file, use the following code:

```
from azureml.core.model import Model

model = Model.register(model_path = "model.pkl",
                      model_name = "Mymodel",
                      tags = ["0.1"],
                      description = "test",
                      workspace = ws)
```

NOTE

While the example shows using a model stored as a pickle file, you can also use ONNX models. For more information on using ONNX models, see the [ONNX and Azure Machine Learning](#) document.

For more information, see the reference documentation for the [Model class](#).

Create an image configuration

Deployed models are packaged as an image. The image contains the dependencies needed to run the model.

For **Azure Container Instance**, **Azure Kubernetes Service**, and **Azure IoT Edge** deployments, the `azureml.core.image.ContainerImage` class is used to create an image configuration. The image configuration is then used to create a new Docker image.

The following code demonstrates how to create a new image configuration:

```
from azureml.core.image import ContainerImage

# Image configuration
image_config = ContainerImage.image_configuration(execution_script = "score.py",
                                                   runtime = "python",
                                                   conda_file = "myenv.yml",
                                                   description = "Image with ridge regression model",
                                                   tags = {"data": "diabetes", "type": "regression"})
```

This configuration uses a `score.py` file to pass requests to the model. This file contains two functions:

- `init()` : Typically this function loads the model into a global object. This function is run only once when the Docker container is started.
- `run(input_data)` : This 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.

For an example `score.py` file, see the [image classification tutorial](#). For an example that uses an ONNX model, see the [ONNX and Azure Machine Learning](#) document.

The `conda_file` parameter is used to provide a conda environment file. This file defines the conda environment for the deployed model. For more information on creating this file, see [Create an environment file \(myenv.yml\)](#).

For more information, see the reference documentation for [ContainerImage class](#)

Create the image

Once you have created the image configuration, you can use it to create an image. This image is stored in the container registry for your workspace. Once created, you can deploy the same image to multiple services.

```
# Create the image from the image configuration
image = ContainerImage.create(name = "myimage",
                               models = [model], #this is the model object
                               image_config = image_config,
                               workspace = ws
                             )
```

Time estimate: Approximately 3 minutes.

Images are versioned automatically when you register multiple images with the same name. For example, the first image registered as `myimage` is assigned an ID of `myimage:1`. The next time you register an image as `myimage`, the ID of the new image is `myimage:2`.

For more information, see the reference documentation for [ContainerImage class](#).

Deploy the image

When you get to deployment, the process is slightly different depending on the compute target that you deploy to. Use the information in the following sections to learn how to deploy to:

- [Azure Container Instances](#)
- [Azure Kubernetes Services](#)
- [Project Brainwave \(field-programmable gate arrays\)](#)
- [Azure IoT Edge devices](#)

Deploy to Azure Container Instances

Use Azure Container Instances for deploying your models as a web service if one or more of the following conditions is true:

- You need to quickly deploy and validate your model. ACI deployment is finished in less than 5 minutes.
- You are testing a model that is under development. To see quota and region availability for ACI, see the [Quotas and region availability for Azure Container Instances](#) document.

To deploy to Azure Container Instances, use the following steps:

1. Define the deployment configuration. The following example defines a configuration that uses one

CPU core and 1 GB of memory:

```
from azureml.core.webservice import AciWebservice

aciconfig = AciWebservice.deploy_configuration(cpu_cores = 1,
                                              memory_gb = 1,
                                              tags = {"data": "mnist", "type": "classification"},
                                              description = 'Handwriting recognition')
```

2. To deploy the image created in the [Create the image](#) section of this document, use the following code:

```
from azureml.core.webservice import Webservice

service_name = 'aci-mnist-13'
service = Webservice.deploy_from_image(deployment_config = aciconfig,
                                       image = image,
                                       name = service_name,
                                       workspace = ws)
service.wait_for_deployment(show_output = True)
print(service.state)
```

Time estimate: Approximately 3 minutes.

TIP

If there are errors during deployment, use `service.get_logs()` to view the AKS service logs. The logged information may indicate the cause of the error.

For more information, see the reference documentation for the [AciWebService](#) and [Webservice](#) classes.

Deploy to Azure Kubernetes Service

To deploy your model as a high-scale production web service, use Azure Kubernetes Service (AKS). You can use an existing AKS cluster or create a new one using the Azure Machine Learning SDK, CLI, or the Azure portal.

Creating an AKS cluster is a one time process for your workspace. You can reuse this cluster for multiple deployments. If you delete the cluster, then you must create a new cluster the next time you need to deploy.

Azure Kubernetes Service provides the following capabilities:

- Autoscaling
- Logging
- Model data collection
- Fast response times for your web services

To deploy to Azure Kubernetes Service, use the following steps:

1. To create an AKS cluster, use the following code:

IMPORTANT

Creating the AKS cluster is a one time process for your workspace. Once created, you can reuse this cluster for multiple deployments. If you delete the cluster or the resource group that contains it, then you must create a new cluster the next time you need to deploy.

```

from azureml.core.compute import AksCompute, ComputeTarget

# Use the default configuration (you can also provide parameters to customize this)
prov_config = AksCompute.provisioning_configuration()

aks_name = 'aml-aks-1'
# Create the cluster
aks_target = ComputeTarget.create(workspace = ws,
                                   name = aks_name,
                                   provisioning_configuration = prov_config)

# Wait for the create process to complete
aks_target.wait_for_completion(show_output = True)
print(aks_target.provisioning_state)
print(aks_target.provisioning_errors)

```

Time estimate: Approximately 20 minutes.

TIP

If you already have AKS cluster in your Azure subscription, and it is version 1.11.*, you can use it to deploy your image. The following code demonstrates how to attach an existing cluster to your workspace:

```

# Set the resource group that contains the AKS cluster and the cluster name
resource_group = 'myresourcegroup'
cluster_name = 'mycluster'

# Attach the cluster to your workgroup
attach_config = AksCompute.attach_configuration(resource_group = resource_group,
                                                 cluster_name = cluster_name)
compute = ComputeTarget.attach(ws, 'mycompute', attach_config)

# Wait for the operation to complete
aks_target.wait_for_completion(True)

```

2. To deploy the image created in the [Create the image](#) section of this document, use the following code:

```

from azureml.core.webservice import Webservice, AksWebservice

# Set configuration and service name
aks_config = AksWebservice.deploy_configuration()
aks_service_name = 'aks-service-1'
# Deploy from image
service = Webservice.deploy_from_image(workspace = ws,
                                         name = aks_service_name,
                                         image = image,
                                         deployment_config = aks_config,
                                         deployment_target = aks_target)

# Wait for the deployment to complete
service.wait_for_deployment(show_output = True)
print(service.state)

```

TIP

If there are errors during deployment, use `service.get_logs()` to view the AKS service logs. The logged information may indicate the cause of the error.

For more information, see the reference documentation for the [AksWebservice](#) and [Webservice](#) classes.

Deploy to field-programmable gate arrays (FPGA)

Project Brainwave makes it possible to achieve ultra-low latency for real-time inferencing requests. Project Brainwave accelerates deep neural networks (DNN) deployed on field-programmable gate arrays in the Azure cloud. Commonly used DNNs are available as featurizers for transfer learning, or customizable with weights trained from your own data.

For a walkthrough of deploying a model using Project Brainwave, see the [Deploy to a FPGA](#) document.

Deploy to Azure IoT Edge

An Azure IoT Edge device is a Linux or Windows-based device that runs the Azure IoT Edge runtime. Machine learning models can be deployed to these devices as IoT Edge modules. Deploying a model to an IoT Edge device allows the device to use the model directly, instead of having to send data to the cloud for processing. You get faster response times and less data transfer.

Azure IoT Edge modules are deployed to your device from a container registry. When you create an image from your model, it is stored in the container registry for your workspace.

Set up your environment

- A development environment. For more information, see the [How to configure a development environment](#) document.
- An [Azure IoT Hub](#) in your Azure subscription.
- A trained model. For an example of how to train a model, see the [Train an image classification model with Azure Machine Learning](#) document. A pre-trained model is available on the [AI Toolkit for Azure IoT Edge GitHub repo](#).

Prepare the IoT device

You must create an IoT hub and register a device or reuse one you have with [this script](#).

```
ssh <yourusername>@<yourdeviceip>
sudo wget https://raw.githubusercontent.com/Azure/ai-toolkit-iot-edge/master/amliotedge/createNregister
sudo chmod +x createNregister
sudo ./createNregister <The Azure subscriptionID you want to use> <Resourcegroup to use or create for the
IoT hub> <Azure location to use e.g. eastus2> <the Hub ID you want to use or create> <the device ID you
want to create>
```

Save the resulting connection string after "cs":"{copy this string}".

Initialize your device by downloading [this script](#) into an UbuntuX64 IoT edge node or DSVM to run the following commands:

```
ssh <yourusername>@<yourdeviceip>
sudo wget https://raw.githubusercontent.com/Azure/ai-toolkit-iot-edge/master/amliotedge/installIoTEdge
sudo chmod +x installIoTEdge
sudo ./installIoTEdge
```

The IoT Edge node is ready to receive the connection string for your IoT Hub. Look for the line `device_connection_string:` and paste the connection string from above in between the quotes.

You can also learn how to register your device and install the IoT runtime step by step by following the [Quickstart: Deploy your first IoT Edge module to a Linux x64 device](#) document.

Get the container registry credentials

To deploy an IoT Edge module to your device, Azure IoT needs the credentials for the container registry that Azure Machine Learning service stores docker images in.

You can easily retrieve the necessary container registry credentials in two ways:

- **In the Azure Portal:**

1. Sign in to the [Azure portal](#).
2. Go to your Azure Machine Learning service workspace and select **Overview**. To go to the container registry settings, select the **Registry** link.

myworkspace
Machine Learning Workspace - PREVIEW

Overview (highlighted)
Activity log
Access control (IAM)
Tags
Diagnose and solve problems

Resource group mygroup	Storage myworstoragenwhzkzka
Location East US 2	Registry myworacrylahgrf
Subscription documentationteam	Key Vault myworkeyvaulttmvmuwhc
Subscription ID	Application Insights myworinsightsxztlsmu

3. Once in the container registry, select **Access Keys** and then enable the admin user.

myworacrylahgrf - Access keys
Container registry

Registry name: myworacrylahgrf

Login server: [empty]

Admin user: **Enable** (highlighted)

Username: [empty]

NAME	PASSWORD
password	[empty]
password2	[empty]

4. Save the values for **login server**, **username**, and **password**.

- **With a Python script:**

1. Use the following Python script after the code you ran above to create a container:

```
# Getting your container details
container_reg = ws.get_details()["containerRegistry"]
reg_name=container_reg.split("/")[-1]
container_url = "://" + image.image_location + "\",
subscription_id = ws.subscription_id
from azure.mgmt.containerregistry import ContainerRegistryManagementClient
from azure.mgmt import containerregistry
client = ContainerRegistryManagementClient(ws._auth,subscription_id)
result= client.registries.list_credentials(resource_group_name, reg_name,
custom_headers=None, raw=False)
username = result.username
password = result.passwords[0].value
print('ContainerURL{}'.format(image.image_location))
print('Servername: {}'.format(reg_name))
print('Username: {}'.format(username))
print('Password: {}'.format(password))
```

2. Save the values for ContainerURL, servername, username, and password.

These credentials are necessary to provide the IoT Edge device access to images in your private container registry.

Deploy the model to the device

You can easily deploy a model by running [this script](#) and providing the following information from the steps above: container registry Name, username, password, image location url, desired deployment name, IoT Hub name, and the device ID you created. You can do this in the VM by following these steps:

```
wget https://raw.githubusercontent.com/Azure/ai-toolkit-iot-edge/master/amliotedge/deploymodel  
sudo chmod +x deploymodel  
sudo ./deploymodel <ContainerRegistryName> <username> <password> <imageLocationURL> <DeploymentID>  
<IoTHubname> <DeviceID>
```

Alternatively, you can follow the steps in the [Deploy Azure IoT Edge modules from the Azure portal](#) document to deploy the image to your device. When configuring the **Registry settings** for the device, use the **login server, username, and password** for your workspace container registry.

NOTE

If you're unfamiliar with Azure IoT, see the following documents for information on getting started with the service:

- [Quickstart: Deploy your first IoT Edge module to a Linux device](#)
- [Quickstart: Deploy your first IoT Edge module to a Windows device](#)

Testing web service deployments

To test a web service deployment, you can use the `run` method of the `Webservice` object. In the following example, a JSON document is set to a web service and the result is displayed. The data sent must match what the model expects. In this example, the data format matches the input expected by the diabetes model.

```
import json  
  
test_sample = json.dumps({'data': [  
    [1,2,3,4,5,6,7,8,9,10],  
    [10,9,8,7,6,5,4,3,2,1]  
]})  
test_sample = bytes(test_sample,encoding = 'utf8')  
  
prediction = service.run(input_data = test_sample)  
print(prediction)
```

Update the web service

To update the web service, use the `update` method. The following code demonstrates how to update the web service to use a new image:

```
from azureml.core.webservice import Webservice  
  
service_name = 'aci-mnist-3'  
# Retrieve existing service  
service = Webservice(name = service_name, workspace = ws)  
# Update the image used by the service  
service.update(image = new_image)  
print(service.state)
```

NOTE

When you update an image, the web service is not automatically updated. You must manually update each service that you want to use the new image.

Clean up

To delete a deployed web service, use `service.delete()`.

To delete an image, use `image.delete()`.

To delete a registered model, use `model.delete()`.

Next steps

- [Secure Azure Machine Learning web services with SSL](#)
- [Consume a ML Model deployed as a web service](#)
- [How to run batch predictions](#)

Deploy a model as a web service on an FPGA with Azure Machine Learning service

12/11/2018 • 3 minutes to read • [Edit Online](#)

You can deploy a model as a web service on [field programmable gate arrays \(FPGAs\)](#). Using FPGAs provides ultra-low latency inferencing, even with a single batch size.

Prerequisites

- If you don't have an Azure subscription, create a free account before you begin. Try the [free or paid version of Azure Machine Learning service](#) today.
- An Azure Machine Learning service workspace and the Azure Machine Learning SDK for Python installed. Learn how to get these prerequisites using the [How to configure a development environment](#) document.
 - Your workspace needs to be in the *East US 2* region.
 - Install the contrib extras:

```
pip install --upgrade azureml-sdk[contrib]
```

Create and deploy your model

Create a pipeline to preprocess the input image, featurize it using ResNet 50 on an FPGA, and then run the features through a classifier trained on the ImageNet data set.

Follow the instructions to:

- Define the model pipeline
- Deploy the model
- Consume the deployed model
- Delete deployed services

IMPORTANT

To optimize latency and throughput, your client should be in the same Azure region as the endpoint. Currently the APIs are created in the East US Azure region.

Preprocess image

The first stage of the pipeline is to preprocess the images.

```
import os
import tensorflow as tf

# Input images as a two-dimensional tensor containing an arbitrary number of images represented as strings
import azureml.contrib.brainwave.models.utils as utils
in_images = tf.placeholder(tf.string)
image_tensors = utils.preprocess_array(in_images)
print(image_tensors.shape)
```

Add Featurizer

Initialize the model and download a TensorFlow checkpoint of the quantized version of ResNet50 to be used as a featurizer.

```
from azureml.contrib.brainwave.models import QuantizedResnet50, Resnet50
model_path = os.path.expanduser('~/models')
model = QuantizedResnet50(model_path, is_frozen = True)
feature_tensor = model.import_graph_def(image_tensors)
print(model.version)
print(feature_tensor.name)
print(feature_tensor.shape)
```

Add Classifier

This classifier has been trained on the ImageNet data set.

```
classifier_input, classifier_output = Resnet50.get_default_classifier(feature_tensor, model_path)
```

Create service definition

Now that you have defined the image preprocessing, featurizer, and classifier that runs on the service, you can create a service definition. The service definition is a set of files generated from the model that is deployed to the FPGA service. The service definition consists of a pipeline. The pipeline is a series of stages that are run in order. TensorFlow stages, Keras stages, and BrainWave stages are supported. The stages are run in order on the service, with the output of each stage input into the subsequent stage.

To create a TensorFlow stage, specify a session containing the graph (in this case default graph is used) and the input and output tensors to this stage. This information is used to save the graph so that it can be run on the service.

```
from azureml.contrib.brainwave.pipeline import ModelDefinition, TensorflowStage, BrainWaveStage

save_path = os.path.expanduser('~/models/save')
model_def_path = os.path.join(save_path, 'service_def.zip')

model_def = ModelDefinition()
with tf.Session() as sess:
    model_def.pipeline.append(TensorflowStage(sess, in_images, image_tensors))
    model_def.pipeline.append(BrainWaveStage(sess, model))
    model_def.pipeline.append(TensorflowStage(sess, classifier_input, classifier_output))
model_def.save(model_def_path)
print(model_def_path)
```

Deploy model

Create a service from the service definition. Your workspace needs to be in the East US 2 location.

```

from azureml.core import Workspace

ws = Workspace.from_config()
print(ws.name, ws.resource_group, ws.location, ws.subscription_id, sep = '\n')

from azureml.core.model import Model
model_name = "resnet-50-rtai"
registered_model = Model.register(ws, model_def_path, model_name)

from azureml.core.webservice import Webservice
from azureml.exceptions import WebServiceException
from azureml.contrib.brainwave import BrainwaveWebservice, BrainwaveImage
service_name = "imagenet-infer"
service = None
try:
    service = Webservice(ws, service_name)
except WebServiceException:
    image_config = BrainwaveImage.image_configuration()
    deployment_config = BrainwaveWebservice.deploy_configuration()
    service = Webservice.deploy_from_model(ws, service_name, [registered_model], image_config,
deployment_config)
    service.wait_for_deployment(True)

```

Test the service

To send an image to the API and test the response, add a mapping from the output class ID to the ImageNet class name.

```

import requests
classes_entries =
requests.get("https://raw.githubusercontent.com/Lasagne/Recipes/master/examples/resnet50/imagenet_classes.txt")
.text.splitlines()

```

Call your service and replace the "your-image.jpg" file name below with an image from your machine.

```

with open('your-image.jpg') as f:
    results = service.run(f)
# map results [class_id] => [confidence]
results = enumerate(results)
# sort results by confidence
sorted_results = sorted(results, key=lambda x: x[1], reverse=True)
# print top 5 results
for top in sorted_results[:5]:
    print(classes_entries[top[0]], 'confidence:', top[1])

```

Clean up service

Delete the service.

```

service.delete()

registered_model.delete()

```

Secure FPGA web services

Azure Machine Learning models running on FPGAs provide SSL support and key-based authentication. This enables you to restrict access to your service and secure data submitted by clients. [Learn how to secure the web service](#).

Next steps

Learn how to [Consume a ML Model deployed as a web service.](#)

Troubleshooting Azure Machine Learning service AKS and ACI deployments

12/7/2018 • 6 minutes to read • [Edit Online](#)

In this article, you will learn how to work around or solve the common Docker deployment errors with Azure Container Instances (ACI) and Azure Kubernetes Service (AKS) using Azure Machine Learning service.

When deploying a model in Azure Machine Learning service, the system performs a number of tasks. This is a complex sequence of events and sometimes issues arise. The deployment tasks are:

1. Register the model in the workspace model registry.
2. Build a Docker image, including:
 - a. Download the registered model from the registry.
 - b. Create a dockerfile, with a Python environment based on the dependencies you specify in the environment yaml file.
 - c. Add your model files and the scoring script you supply in the dockerfile.
 - d. Build a new Docker image using the dockerfile.
 - e. Register the Docker image with the Azure Container Registry associated with the workspace.
3. Deploy the Docker image to Azure Container Instance (ACI) service or to Azure Kubernetes Service (AKS).
4. Start up a new container (or containers) in ACI or AKS.

Learn more about this process in the [Model Management](#) introduction.

Before you begin

If you run into any issue, the first thing to do is to break down the deployment task (previous described) into individual steps to isolate the problem.

This is particularly helpful if you are using the `Webservice.deploy` API, or `Webservice.deploy_from_model` API, since those functions group together the aforementioned steps into a single action. Typically those APIs are quite convenient, but it helps to break up the steps when troubleshooting by replacing them with the below API calls.

1. Register the model. Here's some sample code:

```
# register a model out of a run record
model = best_run.register_model(model_name='my_best_model', model_path='outputs/my_model.pkl')

# or, you can register a file or a folder of files as a model
model = Model.register(model_path='my_model.pkl', model_name='my_best_model', workspace=ws)
```

2. Build the image. Here's some sample code:

```

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

# create the image
image = Image.create(name='myimg', models=[model], image_config=image_config, workspace=ws)

# wait for image creation to finish
image.wait_for_creation(show_output=True)

```

3. Deploy the image as service. Here's some sample code:

```

# configure an ACI-based deployment
aci_config = AciWebservice.deploy_configuration(cpu_cores=1, memory_gb=1)

aci_service = Webservice.deploy_from_image(deployment_config=aci_config,
                                            image=image,
                                            name='mysvc',
                                            workspace=ws)
aci_service.wait_for_deployment(show_output=True)

```

Once you have broken down the deployment process into individual tasks, we can look at some of the most common errors.

Image building fails

If system is unable to build the Docker image, the `image.wait_for_creation()` call fails with some error messages that can offer some clues. You can also find out more details about the errors from the image build log. Below is some sample code showing how to discover the image build log uri.

```

# if you already have the image object handy
print(image.image_build_log_uri)

# if you only know the name of the image (note there might be multiple images with the same name but different
# version number)
print(ws.images()['myimg'].image_build_log_uri)

# list logs for all images in the workspace
for name, img in ws.images().items():
    print (img.name, img.version, img.image_build_log_uri)

```

The image log uri is a SAS URL pointing to a log file stored in your Azure blob storage. Simply copy and paste the uri into a browser window and you can download and view the log file.

Service launch fails

After the image is successfully built, the system attempts to start a container in either ACI or AKS depending on your deployment configuration. It is generally recommended to try an ACI deployment first, since it is a simpler single-container deployment. This way you can then rule out any AKS-specific problem.

As part of container starting-up process, the `init()` function in your scoring script is invoked by the system. If there are uncaught exceptions in the `init()` function, you might see **CrashLoopBackOff** error in the error message. Below are some tips to help you troubleshoot the problem.

Inspect the Docker log

You can print out detailed Docker engine log messages from the service object.

```
# if you already have the service object handy
print(service.get_logs())

# if you only know the name of the service (note there might be multiple services with the same name but
different version number)
print(ws.webservices()['mysvc'].get_logs())
```

Debug the Docker image locally

Some times the Docker log does not emit enough information about what is going wrong. You can go one step further and pull down the built Docker image, start a local container, and debug directly inside the live container interactively. To start a local container, you must have a Docker engine running locally, and it would be a lot easier if you also have [azure-cli](#) installed.

First we need to find out the image location:

```
# print image location
print(image.image_location)
```

The image location has this format: <acr-name>.azurecr.io/<image-name>:<version-number>, such as
myworkspaceacr.azurecr.io/myimage:3 .

Now go to your command-line window. If you have azure-cli installed, you can type the following commands to sign in to the ACR (Azure Container Registry) associated with the workspace where the image is stored.

```
# log on to Azure first if you haven't done so before
$ az login

# make sure you set the right subscription in case you have access to multiple subscriptions
$ az account set -s <subscription_name_or_id>

# now let's log in to the workspace ACR
# note the acr-name is the domain name WITHOUT the ".azurecr.io" postfix
# e.g.: az acr login -n myworkspaceacr
$ az acr login -n <acr-name>
```

If you don't have azure-cli installed, you can use `docker login` command to log into the ACR. But you need to retrieve the user name and password of the ACR from Azure portal first.

Once you have logged in to the ACR, you can pull down the Docker image and start a container locally, and then launch a bash session for debugging by using the `docker run` command:

```
# note the image_id is <acr-name>.azurecr.io/<image-name>:<version-number>
# for example: myworkspaceacr.azurecr.io/myimage:3
$ docker run -it <image_id> /bin/bash
```

Once you launch a bash session the running container, you can find your scoring scripts in the `/var/azureml-app` folder. You can then launch a Python session to debug your scoring scripts.

```
# enter the directory where scoring scripts live
cd /var/azureml-app

# find what Python packages are installed in the python environment
pip freeze

# sanity-check on score.py
# you might want to edit the score.py to trigger init().
# as most of the errors happen in init() when you are trying to load the model.
python score.py
```

In case you need a text editor to modify your scripts, you can install vim, nano, Emacs, or your other favorite editor.

```
# update package index
apt-get update

# install a text editor of your choice
apt-get install vim
apt-get install nano
apt-get install emacs

# launch emacs (for example) to edit score.py
emacs score.py

# exit the container bash shell
exit
```

You can also start up the web service locally and send HTTP traffic to it. The Flask server in the Docker container is running on port 5001. You can map to any other ports available on the host machine.

```
# you can find the scoring API at: http://localhost:8000/score
$ docker run -p 8000:5001 <image_id>
```

Function fails: get_model_path()

Often, in the `init()` function in the scoring script, `Model.get_model_path()` function is called to locate a model file or a folder of model files in the container. This is often a source of failure if the model file or folder cannot be found. The easiest way to debug this error is to run the below Python code in the Container shell:

```
from azureml.core.model import Model
print(Model.get_model_path(model_name='my-best-model'))
```

This would print out the local path (relative to `/var/azureml-app`) in the container where your scoring script is expecting to find the model file or folder. Then you can verify if the file or folder is indeed where it is expected to be.

Function fails: run(input_data)

If the service is successfully deployed, but it crashes when you post data to the scoring endpoint, you can add error catching statement in your `run(input_data)` function so that it returns detailed error message instead. For example:

```
def run(input_data):
    try:
        data = json.loads(input_data)['data']
        data = np.array(data)
        result = model.predict(data)
        return json.dumps({"result": result.tolist()})
    except Exception as e:
        result = str(e)
        # return error message back to the client
        return json.dumps({"error": result})
```

Note: Returning error messages from the `run(input_data)` call should be done for debugging purpose only. It might not be a good idea to do this in a production environment for security reasons.

Next steps

Learn more about deployment:

- [How to deploy and where](#)
- [Tutorial: Train & deploy models](#)

Use SSL to secure web services with Azure Machine Learning service

12/7/2018 • 4 minutes to read • [Edit Online](#)

In this article, you will learn how to secure a web service deployed with the Azure Machine Learning service. You can restrict access to web services and secure the data submitted by clients using secure socket layers (SSL) and key-based authentication.

WARNING

If you do not enable SSL, any user on the internet will be able to make calls to the web service.

SSL encrypts data sent between the client and the web service. It also used by the client to verify the identity of the server. Authentication is only enabled for services that have provided an SSL certificate and key. If you enable SSL, an authentication key is required when accessing the web service.

Whether you deploy a web service enabled with SSL or you enable SSL for existing deployed web service, the steps are the same:

1. Get a domain name.
2. Get an SSL certificate.
3. Deploy or update the web service with the SSL setting enabled.
4. Update your DNS to point to the web service.

There are slight differences when securing web services across the [deployment targets](#).

Get a domain name

If you do not already own a domain name, you can purchase one from a **domain name registrar**. The process differs between registrars, as does the cost. The registrar also provides you with tools for managing the domain name. These tools are used to map a fully qualified domain name (such as www.contoso.com) to the IP address hosting your web service.

Get an SSL certificate

There are many ways to get an SSL certificate. The most common is to purchase one from a **Certificate Authority** (CA). Regardless of where you obtain the certificate, you need the following files:

- A **certificate**. The certificate must contain the full certificate chain, and must be PEM-encoded.
- A **key**. The key must be PEM-encoded.

When requesting a certificate, you must provide the fully qualified domain name (FQDN) of the address you plan to use for the web service. For example, www.contoso.com. The address stamped into the certificate and the address used by the clients are compared when validating the identity of the web service. If the addresses do not match, the clients will receive an error.

TIP

If the Certificate Authority cannot provide the certificate and key as PEM-encoded files, you can use a utility such as [OpenSSL](#) to change the format.

WARNING

Self-signed certificates should be used only for development. They should not be used in production. Self-signed certificates can cause problems in your client applications. For more information, see the documentation for the network libraries used in your client application.

Enable SSL and deploy

To deploy (or re-deploy) the service with SSL enabled, set the `ssl_enabled` parameter to `True`, wherever applicable. Set the `ssl_certificate` parameter to the value of the **certificate** file and the `ssl_key` to the value of the **key** file.

- **Deploy on Azure Kubernetes Service (AKS)**

While provisioning the AKS cluster, provide values for SSL-related parameters as shown in the code snippet:

```
from azureml.core.compute import AksCompute

provisioning_config = AksCompute.provisioning_configuration(ssl_cert_pem_file="cert.pem",
    ssl_key_pem_file="key.pem", ssl_cname="www.contoso.com")
```

- **Deploy on Azure Container Instances (ACI)**

While deploying to ACI, provide values for SSL-related parameters as shown in the code snippet:

```
from azureml.core.webservice import AciWebservice

aci_config = AciWebservice.deploy_configuration(ssl_enabled=True, ssl_cert_pem_file="cert.pem",
    ssl_key_pem_file="key.pem", ssl_cname="www.contoso.com")
```

- **Deploy on Field Programmable Gate Arrays (FPGAs)**

The response of the `create_service` operation contains the IP address of the service. The IP address is used when mapping the DNS name to the IP address of the service. The response also contains a **primary key** and **secondary key** that are used to consume the service. Provide values for SSL-related parameters as shown in the code snippet:

```

from amlrealtimeai import DeploymentClient

subscription_id = "<Your Azure Subscription ID>"
resource_group = "<Your Azure Resource Group Name>"
model_management_account = "<Your AzureML Model Management Account Name>"
location = "eastus2"

model_name = "resnet50-model"
service_name = "quickstart-service"

deployment_client = DeploymentClient(subscription_id, resource_group, model_management_account,
location)

with open('cert.pem','r') as cert_file:
    with open('key.pem','r') as key_file:
        cert = cert_file.read()
        key = key_file.read()
        service = deployment_client.create_service(service_name, model_id, ssl_enabled=True,
ssl_certificate=cert, ssl_key=key)

```

Update your DNS

Next, you must update your DNS to point to the web service.

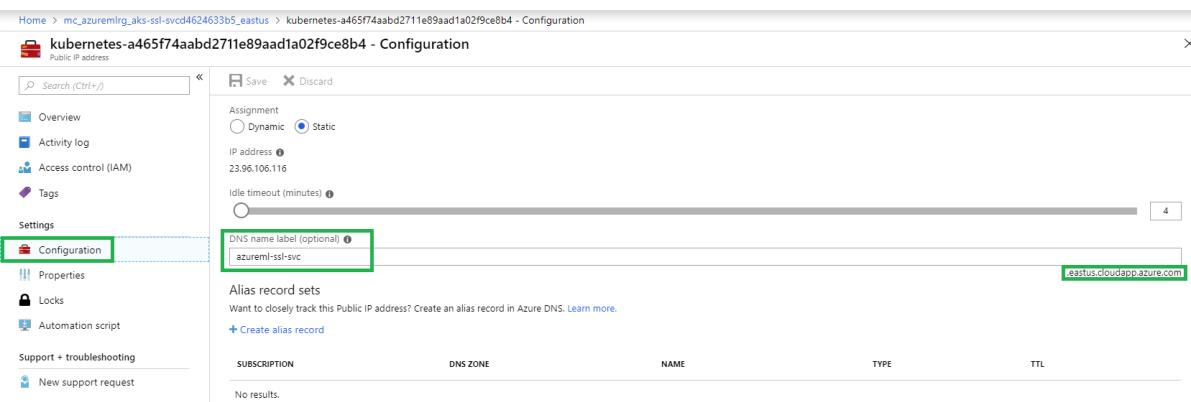
- **For ACI and FPGA:**

Use the tools provided by your domain name registrar to update the DNS record for your domain name. The record must point to the IP address of the service.

Depending on the registrar, and the time to live (TTL) configured for the domain name, it can take several minutes to several hours before clients can resolve the domain name.

- **For AKS:**

Update the DNS under the "Configuration" tab of the "Public IP Address" of the AKS cluster as shown in the image. You can find the Public IP Address as one of the resource types created under the resource group that contains the AKS agent nodes and other networking resources.



Next steps

Learn how to [Consume a ML Model deployed as a web service](#).

Consume an Azure Machine Learning model deployed as a web service

12/11/2018 • 8 minutes to read • [Edit Online](#)

Deploying an Azure Machine Learning model as a web service creates a REST API. You can send data to this API and receive the prediction returned by the model. In this document, learn how to create clients for the web service using C#, Go, Java, and Python.

A web service is created when you deploy an image to an Azure Container Instance, Azure Kubernetes Service, or Project Brainwave (field programmable gate arrays). Images are created from registered models and scoring files. The URI used to access a web service can be retrieved using the [Azure Machine Learning SDK](#). If authentication is enabled, you can also use the SDK to get the authentication keys.

The general workflow when creating a client that uses an ML web service is:

1. Use the SDK to get the connection information
2. Determine the type of request data used by the model
3. Create an application that calls the web service

Connection information

NOTE

The Azure Machine Learning SDK is used to get the web service information. This is a Python SDK. While it is used to retrieve information about the web services, you can use any language to create a client for the service.

The web service connection information can be retrieved using the Azure Machine Learning SDK. The `azureml.core.WebService` class provides the information needed to create a client. The following `WebService` properties that are useful when creating a client application:

- `auth_enabled` - If authentication is enabled, `True`; otherwise, `False`.
- `scoring_uri` - The REST API address.

There are three ways to retrieve this information for deployed web services:

- When you deploy a model, a `WebService` object is returned with information about the service:

```
service = WebService.deploy_from_model(name='myservice',
                                       deployment_config=myconfig,
                                       models=[model],
                                       image_config=image_config,
                                       workspace=ws)
print(service.scoring_uri)
```

- You can use `WebService.list` to retrieve a list of deployed web services for models in your workspace. You can add filters to narrow the list of information returned. For more information on what can be filtered on, see the `WebService.list` reference documentation.

```
services = Webservice.list(ws)
print(services[0].scoring_uri)
```

- If you know the name of the deployed service, you can create a new instance of `Webservice` and provide the workspace and service name as parameters. The new object contains information about the deployed service.

```
service = Webservice(workspace=ws, name='myservice')
print(service.scoring_uri)
```

Authentication key

Authentication keys are created automatically when authentication is enabled for a deployment.

- Authentication is **enabled by default** when deploying to **Azure Kubernetes Service**.
- Authentication is **disabled by default** when deploying to **Azure container Instances**.

To control authentication, use the `auth_enabled` parameter when creating or updating a deployment.

If authentication is enabled, you can use the `get_keys` method to retrieve a primary and secondary authentication key:

```
primary, secondary = service.get_keys()
print(primary)
```

IMPORTANT

If you need to regenerate a key, use `service.regen_key`.

Request data

The REST API expects the body of the request to be a JSON document with the following structure:

```
{
  "data":
    [
      <model-specific-data-structure>
    ]
}
```

IMPORTANT

The structure of the data needs to match what the scoring script and model in the service expect. The scoring script might modify the data before passing it to the model.

For example, the model in the [Train within notebook](#) example expects an array of 10 numbers. The scoring script for this example creates a Numpy array from the request and passes it to the model. The following example shows the data this service expects:

```
{
  "data": [
    [
      [
        0.0199132141783263,
        0.0506801187398187,
        0.104808689473925,
        0.0700725447072635,
        -0.0359677812752396,
        -0.0266789028311707,
        -0.0249926566315915,
        -0.00259226199818282,
        0.00371173823343597,
        0.0403433716478807
      ]
    ]
  }
}
```

The web service can accept multiple sets of data in one request. It returns a JSON document containing an array of responses.

Binary data

If your model accepts binary data, such as an image, you must modify the `score.py` file used for your deployment to accept raw HTTP requests. Here's an example of a `score.py` that accepts binary data and returns the reversed bytes for POST requests. For GET requests it returns the full URL in the response body:

```
from azureml.contrib.services.aml_request import AMLRequest, rawhttp
from azureml.contrib.services.aml_response import AMLResponse

def init():
    print("This is init()")

@rawhttp
def run(request):
    print("This is run()")
    print("Request: [{0}]".format(request))
    if request.method == 'GET':
        respBody = str.encode(request.full_path)
        return AMLResponse(respBody, 200)
    elif request.method == 'POST':
        reqBody = request.get_data(False)
        respBody = bytearray(reqBody)
        respBody.reverse()
        respBody = bytes(respBody)
        return AMLResponse(respBody, 200)
    else:
        return AMLResponse("bad request", 500)
```

IMPORTANT

Things in the `azureml.contrib` namespace change frequently as we work to improve the service. As such, anything in this namespace should be considered as a preview and not fully supported by Microsoft.

If you need to test this on your local development environment, you can install the components in the contrib namespace using the following command:

```
pip install azureml-contrib-services
```

Call the service (C#)

This example demonstrates how to use C# to call the web service created from the [Train within notebook](#) example:

```
using System;
using System.Collections.Generic;
using System.IO;
using System.Net.Http;
using System.Net.Http.Headers;
using Newtonsoft.Json;

namespace MLWebServiceClient
{
    // The data structure expected by the service
    internal class InputData
    {
        [JsonProperty("data")]
        // The service used by this example expects an array containing
        // one or more arrays of doubles
        internal double[,] data;
    }
    class Program
    {
        static void Main(string[] args)
        {
            // Set the scoring URI and authentication key
            string scoringUri = "<your web service URI>";
            string authKey = "<your key>";

            // Set the data to be sent to the service.
            // In this case, we are sending two sets of data to be scored.
            InputData payload = new InputData();
            payload.data = new double[,] {
                {
                    0.0199132141783263,
                    0.0506801187398187,
                    0.104808689473925,
                    0.0700725447072635,
                    -0.0359677812752396,
                    -0.0266789028311707,
                    -0.0249926566315915,
                    -0.00259226199818282,
                    0.00371173823343597,
                    0.0403433716478807
                },
                {
                    -0.0127796318808497,
                    -0.044641636506989,
                    0.0606183944448076,
                    0.0528581912385822,
                    0.0479653430750293,
                    0.0293746718291555,
                    -0.0176293810234174,
                    0.0343088588777263,
                    0.0702112981933102,
                    0.00720651632920303
                }
            };
            // Create the HTTP client
            HttpClient client = new HttpClient();
            // Set the auth header. Only needed if the web service requires authentication.
            client.DefaultRequestHeaders.Authorization = new AuthenticationHeaderValue("Bearer", authKey);

            // Make the request
            try {
```

```
        var request = new HttpRequestMessage(HttpMethod.Post, new Uri(scoringUri));
        request.Content = new StringContent(JsonConvert.SerializeObject(payload));
        request.Content.Headers.ContentType = new MediaTypeHeaderValue("application/json");
        var response = client.SendAsync(request).Result;
        // Display the response from the web service
        Console.WriteLine(response.Content.ReadAsStringAsync().Result);
    }
    catch (Exception e)
    {
        Console.Out.WriteLine(e.Message);
    }
}
}
```

The results returned are similar to the following JSON document:

[217.67978776218715, 224.78937091757172]

Call the service (Go)

This example demonstrates how to use Go to call the web service created from the [Train within notebook](#) example:

```
package main

import (
    "bytes"
    "encoding/json"
    "fmt"
    "io/ioutil"
    "net/http"
)

// Features for this model are an array of decimal values
type Features []float64

// The web service input can accept multiple sets of values for scoring
type InputData struct {
    Data []Features `json:"data",omitempty`
}

// Define some example data
var exampleData = []Features{
    []float64{
        0.0199132141783263,
        0.0506801187398187,
        0.104808689473925,
        0.0700725447072635,
        -0.0359677812752396,
        -0.0266789028311707,
        -0.0249926566315915,
        -0.00259226199818282,
        0.00371173823343597,
        0.0403433716478807,
    },
    []float64{
        -0.0127796318808497,
        -0.044641636506989,
        0.0606183944448076,
        0.0528581912385822,
        0.0479653430750293,
        0.0293746718291555,
        -0.0176293810234174
    }
}
```

```

    0.0343088588777263,
    0.0702112981933102,
    0.00720651632920303,
},
}

// Set to the URI for your service
var serviceUri string = "<your web service URI>"
// Set to the authentication key (if any) for your service
var authKey string = "<your key>"


func main() {
    // Create the input data from example data
    jsonData := InputData{
        Data: exampleData,
    }
    // Create JSON from it and create the body for the HTTP request
    jsonValue, _ := json.Marshal(jsonData)
    body := bytes.NewBuffer(jsonValue)

    // Create the HTTP request
    client := &http.Client{}
    request, err := http.NewRequest("POST", serviceUri, body)
    request.Header.Add("Content-Type", "application/json")

    // These next two are only needed if using an authentication key
    bearer := fmt.Sprintf("Bearer %v", authKey)
    request.Header.Add("Authorization", bearer)

    // Send the request to the web service
    resp, err := client.Do(request)
    if err != nil {
        fmt.Println("Failure: ", err)
    }

    // Display the response received
    respBody, _ := ioutil.ReadAll(resp.Body)
    fmt.Println(string(respBody))
}

```

The results returned are similar to the following JSON document:

```
[217.67978776218715, 224.78937091757172]
```

Call the service (Java)

This example demonstrates how to use Java to call the web service created from the [Train within notebook](#) example:

```

import java.io.IOException;
import org.apache.http.client.fluent.*;
import org.apache.http.entity.ContentType;
import org.json.simple.JSONArray;
import org.json.simple.JSONObject;

public class App {
    // Handle making the request
    public static void sendRequest(String data) {
        // Replace with the scoring_uri of your service
        String uri = "<your web service URI>";
        // If using authentication, replace with the auth key
        String key = "<your key>";
        try {
            // Create the request

```

```

        Content content = Request.Post(uri)
        .addHeader("Content-Type", "application/json")
        // Only needed if using authentication
        .addHeader("Authorization", "Bearer " + key)
        // Set the JSON data as the body
        .bodyString(data, ContentType.APPLICATION_JSON)
        // Make the request and display the response.
        .execute().returnContent();
        System.out.println(content);
    }
    catch (IOException e) {
        System.out.println(e);
    }
}

public static void main(String[] args) {
    // Create the data to send to the service
    JSONObject obj = new JSONObject();
    // In this case, it's an array of arrays
    JSONArray dataItems = new JSONArray();
    // Inner array has 10 elements
    JSONArray item1 = new JSONArray();
    item1.add(0.0199132141783263);
    item1.add(0.0506801187398187);
    item1.add(0.104808689473925);
    item1.add(0.0700725447072635);
    item1.add(-0.0359677812752396);
    item1.add(-0.0266789028311707);
    item1.add(-0.0249926566315915);
    item1.add(-0.00259226199818282);
    item1.add(0.00371173823343597);
    item1.add(0.0403433716478807);
    // Add the first set of data to be scored
    dataItems.add(item1);
    // Create and add the second set
    JSONArray item2 = new JSONArray();
    item2.add(-0.0127796318808497);
    item2.add(-0.044641636506989);
    item2.add(0.0606183944448076);
    item2.add(0.0528581912385822);
    item2.add(0.0479653430750293);
    item2.add(0.0293746718291555);
    item2.add(-0.0176293810234174);
    item2.add(0.0343088588777263);
    item2.add(0.0702112981933102);
    item2.add(0.00720651632920303);
    dataItems.add(item2);
    obj.put("data", dataItems);

    // Make the request using the JSON document string
    sendRequest(obj.toJSONString());
}
}

```

The results returned are similar to the following JSON document:

```
[217.67978776218715, 224.78937091757172]
```

Call the service (Python)

This example demonstrates how to use Python to call the web service created from the [Train within notebook](#) example:

```

import requests
import requests
import json

# URL for the web service
scoring_uri = '<your web service URI>'
# If the service is authenticated, set the key
key = '<your key>'

# Two sets of data to score, so we get two results back
data = {"data":
    [
        [
            0.0199132141783263,
            0.0506801187398187,
            0.104808689473925,
            0.0700725447072635,
            -0.0359677812752396,
            -0.0266789028311707,
            -0.0249926566315915,
            -0.00259226199818282,
            0.00371173823343597,
            0.0403433716478807
        ],
        [
            -0.0127796318808497,
            -0.044641636506989,
            0.0606183944448076,
            0.0528581912385822,
            0.0479653430750293,
            0.0293746718291555,
            -0.0176293810234174,
            0.0343088588777263,
            0.0702112981933102,
            0.00720651632920303
        ]
    ]
}

# Convert to JSON string
input_data = json.dumps(data)

# Set the content type
headers = { 'Content-Type':'application/json' }
# If authentication is enabled, set the authorization header
headers['Authorization']=f'Bearer {key}'

# Make the request and display the response
resp = requests.post(scoring_uri, input_data, headers = headers)
print(resp.text)

```

The results returned are similar to the following JSON document:

```
[217.67978776218715, 224.78937091757172]
```

Run batch predictions on large data sets with Azure Machine Learning service

12/11/2018 • 6 minutes to read • [Edit Online](#)

In this article, you'll learn how to quickly and efficiently make predictions on large quantities of data asynchronously using Azure Machine Learning service.

Batch prediction (or batch scoring) provides cost-effective inference with unparalleled throughput for asynchronous applications. Batch prediction pipelines can scale to perform inference on terabytes of production data. Batch prediction is optimized around high throughput, fire-and-forget predictions for a large collection of data.

NOTE

If your system requires low-latency processing (process a single document or small set of documents quickly), use [real-time scoring](#) instead of batch prediction.

In the following steps, you'll create a [machine learning pipeline](#) to register a pretrained computer vision model ([Inception-V3](#)) and then use the pretrained model to do batch scoring on images available in your Azure blob account. These images used for scoring are unlabeled images from the [ImageNet](#) dataset.

Prerequisites

- If you don't have an Azure subscription, create a free account before you begin. Try the [free or paid version of Azure Machine Learning service](#) today.
- Configure your development environment to install the Azure Machine Learning SDK. For more information, see [Configure a development environment for Azure Machine Learning](#).
- Create an Azure Machine Learning workspace that will hold all your pipeline resources. You can use the following code, or for more options, see [Create a workspace configuration file](#).

```
ws = Workspace.create(  
    name = '<workspace-name>',  
    subscription_id = '<subscription-id>',  
    resource_group = '<resource-group>',  
    location = '<workspace_region>',  
    exist_ok = True)
```

Set up machine learning resources

The following steps will set up the resources you need to run a pipeline:

- Access the datastore that already has the pretrained model, input labels, and images to score (this is already set up for you).
- Set up a datastore to store your outputs.
- Configure DataReference objects to point to the data in the preceding datastores.
- Set up compute machines or clusters where the pipeline steps will run.

Access the datastores

First, access the datastore that has the model, labels, and images.

You'll use a public blob container named *sampledata* in the *pipelinedata* account that holds images from the ImageNet evaluation set. The datastore name for this public container is *images_datastore*. Register this datastore with your workspace:

```
# Public blob container details
account_name = "pipelinedata"
datastore_name="images_datastore"
container_name="sampledata"

batchscore_blob = Datastore.register_azure_blob_container(ws,
    datastore_name=datastore_name,
    container_name= container_name,
    account_name=account_name,
    overwrite=True)
```

Next, setup to use the default datastore for the outputs.

When you create your workspace, an [Azure filestorage](#) and a [blob storage](#) are attached to the workspace by default. Azure file storage is the "default datastore" for a workspace, but you can also use blob storage as a datastore. Learn more about [Azure storage options](#).

```
def_data_store = ws.get_default_datastore()
```

Configure data references

Now, reference the data in your pipeline as inputs to pipeline steps.

A data source in a pipeline is represented by a [DataReference](#) object. The DataReference object points to data that lives in or is accessible from a datastore. You need DataReference objects for the directory used for input images, the directory in which the pretrained model is stored, the directory for labels, and the output directory.

```
input_images = DataReference(datastore=batchscore_blob,
    data_reference_name="input_images",
    path_on_datastore="batchscoring/images",
    mode="download")

model_dir = DataReference(datastore=batchscore_blob,
    data_reference_name="input_model",
    path_on_datastore="batchscoring/models",
    mode="download")

label_dir = DataReference(datastore=batchscore_blob,
    data_reference_name="input_labels",
    path_on_datastore="batchscoring/labels",
    mode="download")

output_dir = PipelineData(name="scores",
    datastore=def_data_store,
    output_path_on_compute="batchscoring/results")
```

Set up compute target

In Azure Machine Learning, compute (or compute target) refers to the machines or clusters that will perform the computational steps in your machine learning pipeline. For example, you can create an

[Azure Machine Learning compute](#).

```

compute_name = "gpucluster"
compute_min_nodes = 0
compute_max_nodes = 4
vm_size = "STANDARD_NC6"

if compute_name in ws.compute_targets:
    compute_target = ws.compute_targets[compute_name]
    if compute_target and type(compute_target) is AmlCompute:
        print('Found compute target. just use it. ' + compute_name)
else:
    print('Creating a new compute target...')
    provisioning_config = AmlCompute.provisioning_configuration(
        vm_size = vm_size, # NC6 is GPU-enabled
        vm_priority = 'lowpriority', # optional
        min_nodes = compute_min_nodes,
        max_nodes = compute_max_nodes)

# create the cluster
compute_target = ComputeTarget.create(ws,
                                       compute_name,
                                       provisioning_config)

compute_target.wait_for_completion(
    show_output=True,
    min_node_count=None,
    timeout_in_minutes=20)

```

Prepare the model

Before you can use the pretrained model, you'll need to download the model and register it with your workspace.

Download the pretrained model

Download the pretrained computer vision model (InceptionV3) from

http://download.tensorflow.org/models/inception_v3_2016_08_28.tar.gz. Once downloaded, extract it to the `models` subfolder.

```

import os
import tarfile
import urllib.request

model_dir = 'models'
if not os.path.isdir(model_dir):
    os.mkdir(model_dir)

url="http://download.tensorflow.org/models/inception_v3_2016_08_28.tar.gz"
response = urllib.request.urlretrieve(url, "model.tar.gz")
tar = tarfile.open("model.tar.gz", "r:gz")
tar.extractall(model_dir)

```

Register the model

```
import shutil
from azureml.core.model import Model

# register downloaded model
model = Model.register(
    model_path = "models/inception_v3.ckpt",
    model_name = "inception", # This is the name of the registered model
    tags = {'pretrained': "inception"},
    description = "Imagenet trained tensorflow inception",
    workspace = ws)
```

Write your scoring script

WARNING

The following code is only a sample of what is contained in the `batch_score.py` used by the [sample notebook](#). You'll need to create your own scoring script for your scenario.

The `batch_score.py` script takes input images in `dataset_path`, pretrained models in `model_dir`, and outputs `results-label.txt` to `output_dir`.

```

# Snippets from a sample scoring script
# Refer to the accompanying batch-scoring Notebook
# https://github.com/Azure/MachineLearningNotebooks/blob/master/pipeline/pipeline-batch-scoring.ipynb
# for the implementation script

# Get labels
def get_class_label_dict(label_file):
    label = []
    proto_as_ascii_lines = tf.gfile.GFile(label_file).readlines()
    for l in proto_as_ascii_lines:
        label.append(l.rstrip())
    return label

class DataIterator:
    # Definition of the DataIterator here

def main(_):
    # Refer to batch-scoring Notebook for implementation.
    label_file_name = os.path.join(args.label_dir, "labels.txt")
    label_dict = get_class_label_dict(label_file_name)
    classes_num = len(label_dict)
    test_feeder = DataIterator(data_dir=args.dataset_path)
    total_size = len(test_feeder.labels)

    # get model from model registry
    model_path = Model.get_model_path(args.model_name)
    with tf.Session() as sess:
        test_images = test_feeder.input_pipeline(batch_size=args.batch_size)
        with slim.arg_scope(inception_v3.inception_v3_arg_scope()):
            input_images = tf.placeholder(tf.float32, [args.batch_size, image_size, image_size, num_channel])
            logits, _ = inception_v3.inception_v3(input_images,
                                                   num_classes=classes_num,
                                                   is_training=False)
        probabilities = tf.argmax(logits, 1)

        sess.run(tf.global_variables_initializer())
        sess.run(tf.local_variables_initializer())
        coord = tf.train.Coordinator()
        threads = tf.train.start_queue_runners(sess=sess, coord=coord)
        saver = tf.train.Saver()
        saver.restore(sess, model_path)
        out_filename = os.path.join(args.output_dir, "result-labels.txt")

    # copy the file to artifacts
    shutil.copy(out_filename, "./outputs/")

```

Build and run the batch scoring pipeline

You have everything you need to build the pipeline, so now put it all together.

Prepare the run environment

Specify the conda dependencies for your script. You'll need this object when you create the pipeline step later.

```

from azureml.core.runconfig import DEFAULT_GPU_IMAGE

cd = CondaDependencies.create(pip_packages=["tensorflow-gpu==1.10.0", "azureml-defaults"])

# Runconfig
amlcompute_run_config = RunConfiguration(conda_dependencies=cd)
amlcompute_run_config.environment.docker.enabled = True
amlcompute_run_config.environment.docker.gpu_support = True
amlcompute_run_config.environment.docker.base_image = DEFAULT_GPU_IMAGE
amlcompute_run_config.environment.spark.precache_packages = False

```

Specify the parameter for your pipeline

Create a pipeline parameter using a [PipelineParameter](#) object with a default value.

```
batch_size_param = PipelineParameter(  
    name="param_batch_size",  
    default_value=20)
```

Create the pipeline step

Create the pipeline step using the script, environment configuration, and parameters. Specify the compute target you already attached to your workspace as the target of execution of the script. Use [PythonScriptStep](#) to create the pipeline step.

```
inception_model_name = "inception_v3.ckpt"  
  
batch_score_step = PythonScriptStep(  
    name="batch_scoring",  
    script_name="batch_score.py",  
    arguments=[>--dataset_path", input_images,  
              "--model_name", "inception",  
              "--label_dir", label_dir,  
              "--output_dir", output_dir,  
              "--batch_size", batch_size_param],  
    compute_target=compute_target,  
    inputs=[input_images, label_dir],  
    outputs=[output_dir],  
    runconfig=amlcompute_run_config,  
    source_directory=scripts_folder
```

Run the pipeline

Now run the pipeline and examine the output it produced. The output will have a score corresponding to each input image.

```
# Run the pipeline  
pipeline = Pipeline(workspace=ws, steps=[batch_score_step])  
pipeline_run = Experiment(ws, 'batch_scoring').submit(pipeline, pipeline_params={"param_batch_size": 20})  
  
# Wait for the run to finish (this may take several minutes)  
pipeline_run.wait_for_completion(show_output=True)  
  
# Download and review the output  
step_run = list(pipeline_run.get_children())[0]  
step_run.download_file("./outputs/result-labels.txt")  
  
import pandas as pd  
df = pd.read_csv("result-labels.txt", delimiter=":", header=None)  
df.columns = ["Filename", "Prediction"]  
df.head()
```

Publish the pipeline

Once you're satisfied with the outcome of the run, publish the pipeline so you can run it with different input values later. When you publish a pipeline, you get a REST endpoint that accepts invoking of the pipeline with the set of parameters you have already incorporated using [PipelineParameter](#).

```
published_pipeline = pipeline_run.publish_pipeline(  
    name="Inception_v3_scoring",  
    description="Batch scoring using Inception v3 model",  
    version="1.0")
```

Rerun the pipeline using the REST endpoint

To rerun the pipeline, you'll need an Azure Active Directory authentication header token as described in [AzureCliAuthentication class](#).

```
from azureml.pipeline.core import PublishedPipeline  
  
rest_endpoint = published_pipeline.endpoint  
# specify batch size when running the pipeline  
response = requests.post(rest_endpoint,  
    headers=aad_token,  
    json={"ExperimentName": "batch_scoring",  
          "ParameterAssignments": {"param_batch_size": 50}})  
  
# Monitor the run  
from azureml.pipeline.core.run import PipelineRun  
published_pipeline_run = PipelineRun(ws.experiments["batch_scoring"], run_id)  
  
RunDetails(published_pipeline_run).show()
```

Next steps

To see this working end-to-end, try the batch scoring notebook in ([how-to-use-azureml/machine-learning-pipelines](#)).

Learn how to run notebooks by following the article, [Use Jupyter notebooks to explore this service](#).

Collect data for models in production

12/11/2018 • 4 minutes to read • [Edit Online](#)

In this article, you can learn how to collect input model data from the Azure Machine Learning services you've deployed into Azure Kubernetes Cluster (AKS) into an Azure Blob storage.

Once enabled, this data you collect helps you:

- Monitor data drifts as production data enters your model
- Make better decisions on when to retrain or optimize your model
- Retrain your model with the data collected

What is collected and where does it go?

The following data can be collected:

- Model **input** data from web services deployed in Azure Kubernetes Cluster (AKS) (Voice, images, and video are **not** collected)
- Model predictions using production input data.

NOTE

Pre-aggregation or pre-calculations on this data are not part of the service at this time.

The output gets saved in an Azure Blob. Since the data gets added into an Azure Blob, you can then choose your favorite tool to run the analysis.

The path to the output data in the blob follows this syntax:

```
/modeldata/<subscriptionid>/<resourcegroup>/<workspace>/<webservice>/<model>/<version>/<identifier>/<year>/<month>/<day>/data.csv  
# example: /modeldata/1a2b3c4d-5e6f-7g8h-9i10-j11k12l13m14/myresourcegrp/myWorkspace/aks-w-colly9/best_model/10/inputs/2018/12/31/data.csv
```

Prerequisites

- If you don't have an Azure subscription, create a free account before you begin. Try the [free or paid version of Azure Machine Learning service](#) today.
- An Azure Machine Learning service workspace, a local directory containing your scripts, and the Azure Machine Learning SDK for Python installed. Learn how to get these prerequisites using the [How to configure a development environment](#) document.
- A trained machine learning model to be deployed to Azure Kubernetes Service (AKS). If you don't have one, see the [train image classification model](#) tutorial.
- An Azure Kubernetes Service cluster. For information on how to create and deploy to one, see the [How to deploy and where](#) document.
- [Set up your environment](#) and install the [Monitoring SDK](#).

Enable data collection

Data collection can be enabled regardless of the model being deployed through Azure Machine Learning Service or other tools.

To enable it, you need to:

1. Open the scoring file.
2. Add the [following code](#) at the top of the file:

```
from azureml.monitoring import ModelDataCollector
```

3. Declare your data collection variables in your `init()` function:

```
global inputs_dc, prediction_dc
inputs_dc = ModelDataCollector("best_model", identifier="inputs", feature_names=["feat1", "feat2",
"feat3", "feat4", "feat5", "feat6"])
prediction_dc = ModelDataCollector("best_model", identifier="predictions", feature_names=["prediction1",
"prediction2"])
```

CorrelationId is an optional parameter, you do not need to set it up if your model doesn't require it. Having a correlationId in place does help you for easier mapping with other data. (Examples include: LoanNumber, CustomerId, etc.)

Identifier is later used for building the folder structure in your Blob, it can be used to divide "raw" data versus "processed".

4. Add the following lines of code to the `run(input_df)` function:

```
data = np.array(data)
result = model.predict(data)
inputs_dc.collect(data) #this call is saving our input data into Azure Blob
prediction_dc.collect(result) #this call is saving our input data into Azure Blob
```

5. Data collection is **not** automatically set to **true** when you deploy a service in AKS, so you must update your configuration file such as:

```
aks_config = AksWebservice.deploy_configuration(collect_model_data=True)
```

AppInsights for service monitoring can also be turned on by changing this configuration:

```
aks_config = AksWebservice.deploy_configuration(collect_model_data=True, enable_app_insights=True)
```

6. To create a new image and deploy the service, see the [How to deploy and where](#) document.

If you already have a service with the dependencies installed in your **environment file** and **scoring file**, enable data collection by:

1. Go to [Azure Portal](#).
2. Open your workspace.
3. Go to **Deployments** -> **Select service** -> **Edit**.

Home > Machine Learning Workspaces > doc-ws

doc-ws
Machine Learning workspace

Experiments Compute Models Images Deployments Activities

cvscodebservice

← Back to Deployments  Edit  Delete

Details Models Images

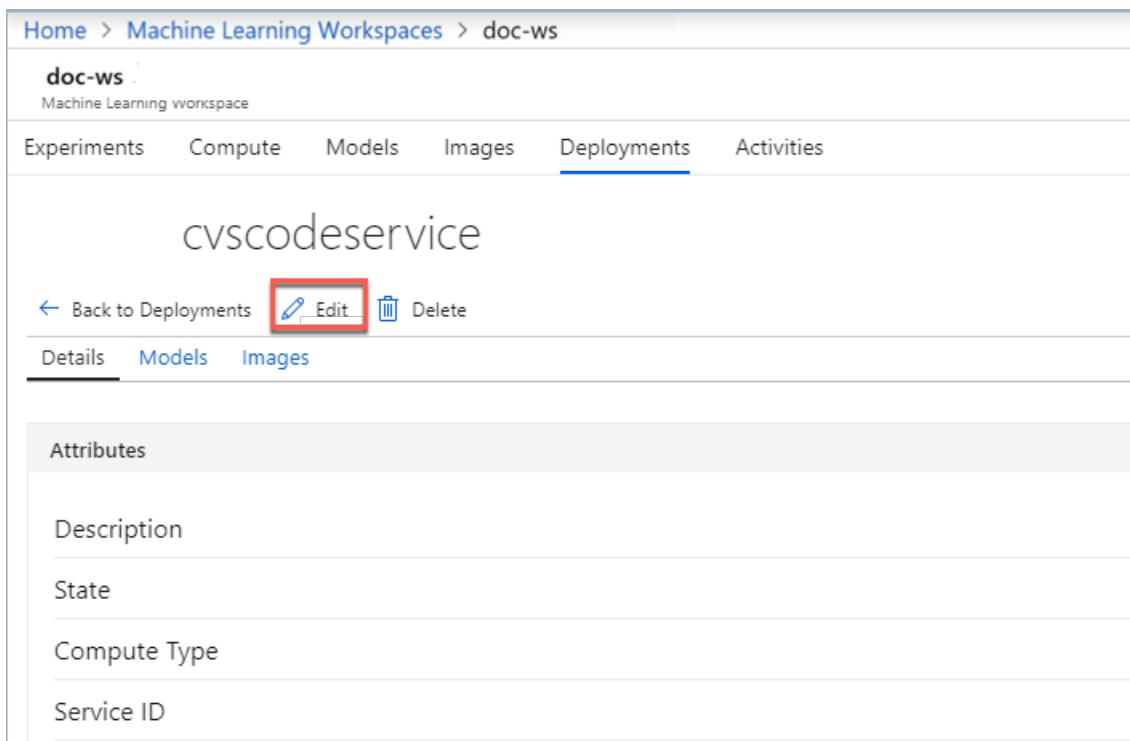
Attributes

Description

State

Compute Type

Service ID



4. In **Advanced Settings**, deselect **Enable Model data collection**.

Home > Machine Learning Workspaces > doc-ws

doc-ws
Machine Learning Workspace

Experiments Compute Models Images Deployments Activities

Update Deployment

* Name
cvscodebservice

Description

Tags
Separate tags with commas

Compute Settings

* Compute Type
AKS

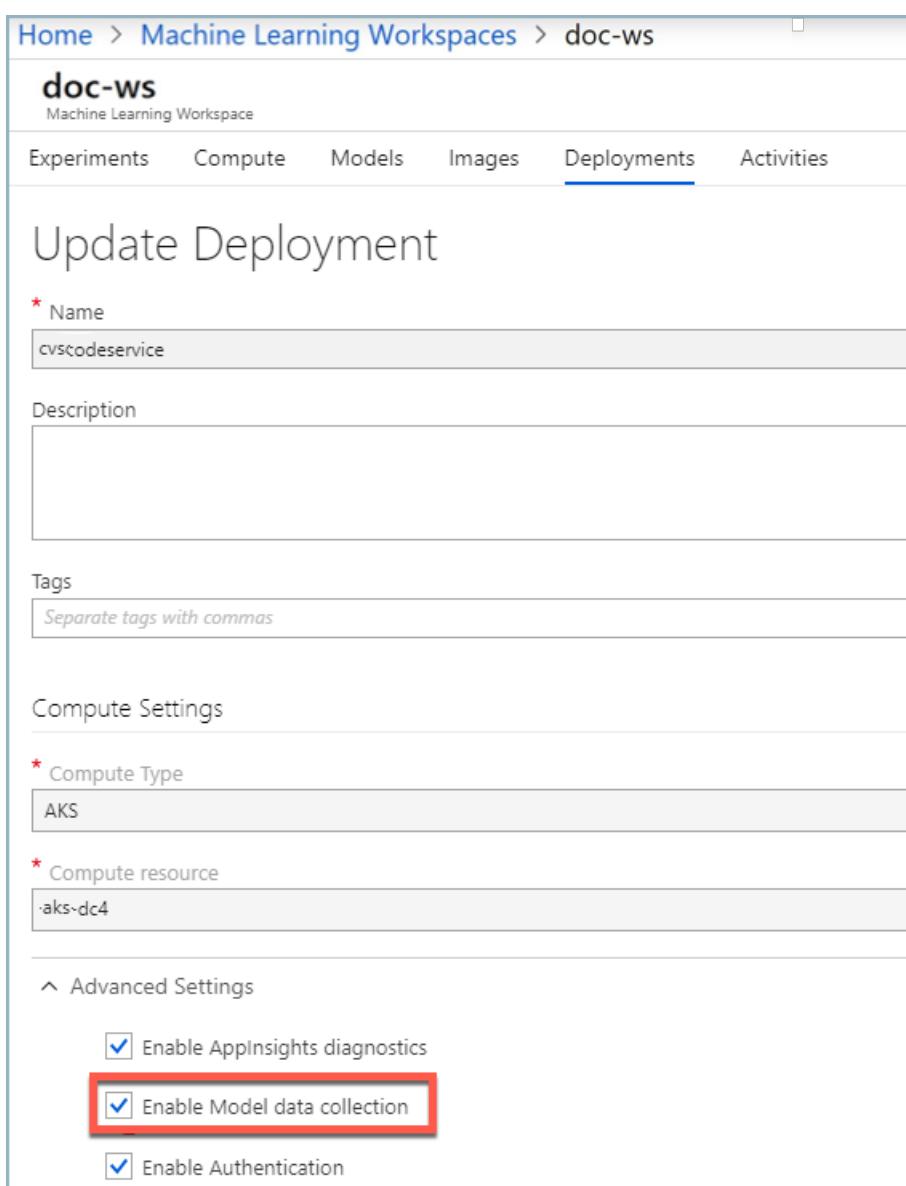
* Compute resource
aks-dc4

Advanced Settings

Enable AppInsights diagnostics

Enable Model data collection

Enable Authentication



In this window, you can also choose to "Enable AppInsights diagnostics" to track the health of your service.

5. Select **Update** to apply the change.

Disable data collection

You can stop collecting data any time. Use Python code or the Azure portal to disable data collection.

- Option 1 - Disable in the Azure portal:

1. Sign in to [Azure portal](#).
2. Open your workspace.
3. Go to **Deployments** -> **Select service** -> **Edit**.

The screenshot shows the Azure Machine Learning Workspaces interface. At the top, there's a breadcrumb navigation: Home > Machine Learning Workspaces > doc-ws. Below that, the workspace name 'doc-ws' is shown with its status as 'Machine Learning workspace'. A navigation bar at the top has tabs for Experiments, Compute, Models, Images, Deployments (which is underlined), and Activities. Under the Deployments tab, a service named 'cvscodeservice' is listed. Below the service name, there are buttons for 'Back to Deployments', 'Edit' (which is highlighted with a red box), and 'Delete'. A sub-navigation bar below these buttons includes 'Details', 'Models', and 'Images', with 'Details' being the active tab. On the left side of the main content area, there's a section titled 'Attributes' which lists 'Description', 'State', 'Compute Type', and 'Service ID'.

4. In **Advanced Settings**, deselect **Enable Model data collection**.

The screenshot shows the 'Advanced Settings' section within the deployment edit screen. It contains three checkboxes: 'Enable AppInsights diagnostics' (checked), 'Enable Model data collection' (unchecked and highlighted with a red box), and 'Enable Authentication' (checked).

5. Select **Update** to apply the change.

- Option 2 - Use Python to disable data collection:

```
## replace <service_name> with the name of the web service
<service_name>.update(collect_model_data=False)
```

Validate your data and analyze it

You can choose any tool of your preference to analyze the data collected into your Azure Blob.

To quickly access the data from your blob:

1. Sign in to [Azure portal](#).

2. Open your workspace.

3. Click on **Storage**.

The screenshot shows the 'My Workspace' page for a Machine Learning service workspace. On the left, there's a sidebar with links like Overview, Activity log, Access control (IAM), Tags, and Diagnose and solve problems. The main area has a 'Delete' button at the top right. Below it, there are sections for Resource group, Location (East US 2), Subscription, and Subscription ID, each with placeholder text. To the right, there's a 'Storage' section with a red box around the 'Storage <blob storage name>' link. Other links in this section include Registry, Key Vault, Application Insights, and App Insights name.

4. Follow the path to the output data in the blob with this syntax:

```
/modeldata/<subscriptionid>/<resourcegroup>/<workspace>/<webservice>/<model>/<version>/<identifier>/<year>/<month>/<day>/data.csv  
# example: /modeldata/1a2b3c4d-5e6f-7g8h-9i10-j11k12l13m14/myresourcegrp/myworkspace/aks-w-collv9/best_model/10/inputs/2018/12/31/data.csv
```

Analyzing model data through Power BI

1. Download and Open [PowerBI Desktop](#)

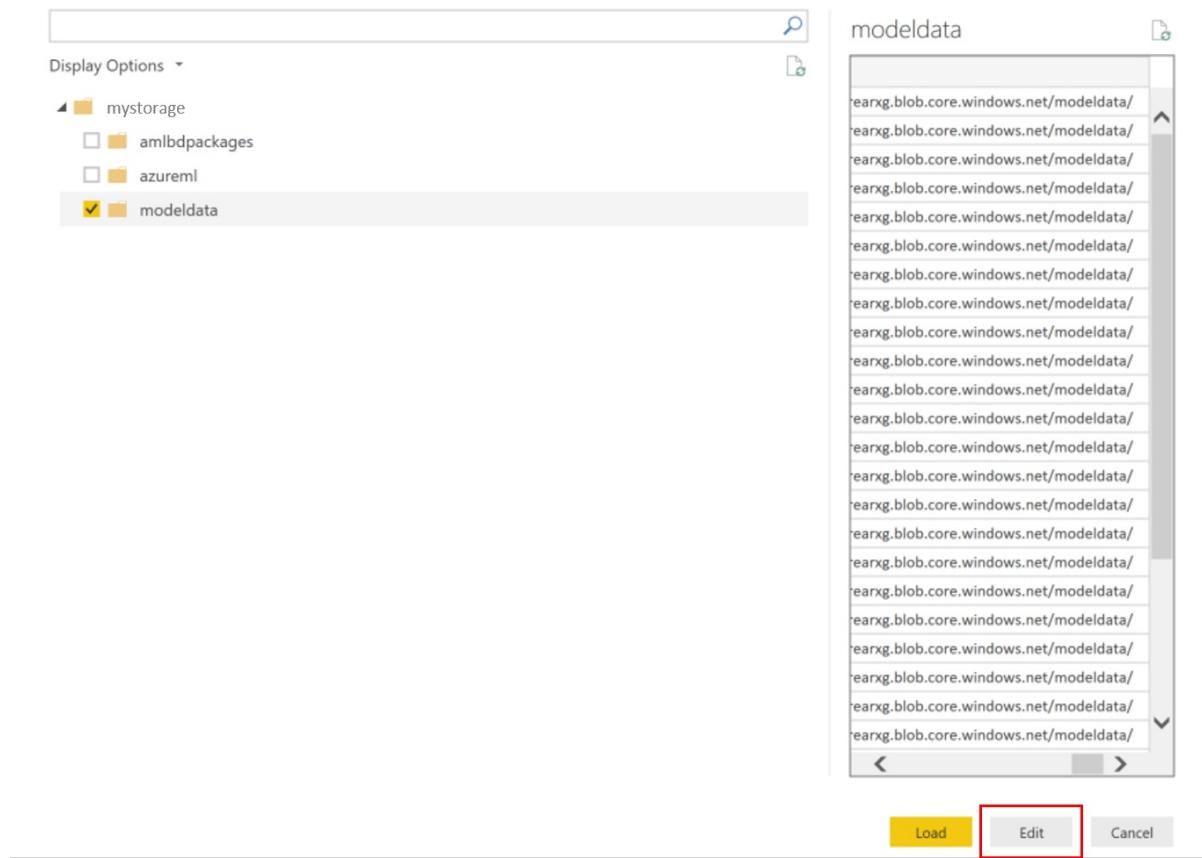
2. Select **Get Data** and click on [Azure Blob Storage](#).

The screenshot shows the 'Get Data' dialog in Power BI. In the search bar, 'azure' is typed. The left sidebar has categories 'All', 'Azure', and 'Online Services'. Under 'All', 'Azure Blob Storage' is highlighted with a yellow bar. Other options include Azure SQL database, Azure SQL Data Warehouse, Azure Analysis Services database, Azure Table Storage, Azure Cosmos DB (Beta), Azure Data Lake Store, Azure HDInsight (HDFS), Azure HDInsight Spark, Microsoft Azure Consumption Insights (Beta), and Azure KustoDB (Beta). At the bottom, there are 'Certified Connectors', a 'Connect' button, and a 'Cancel' button.

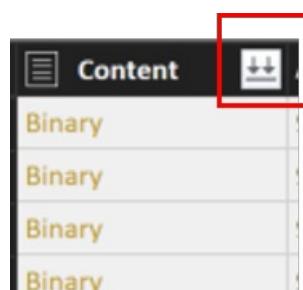
3. Add your storage account name and enter your storage key. You can find this information in your blob's **Settings** >> Access keys.

4. Select the container **modeldata** and click on **Edit**.

Navigator



5. In the query editor, click under "Name" column and add your Storage account 1. Model path into the filter.
Note: if you want to only look into files from a specific year or month, just expand the filter path. For example, just look into March data:
`/modeldata/subscriptionid>/resourcegroupname>/workspacename>/webservicename>/modelname>/modelversion>/identifier>/year>/3`
6. Filter the data that is relevant to you based on **Name**. If you stored **predictions** and **inputs** you'll need to do create a query per each.
7. Click on the double arrow aside the **Content** column to combine the files.



8. Click OK and the data will preload.

Combine Files

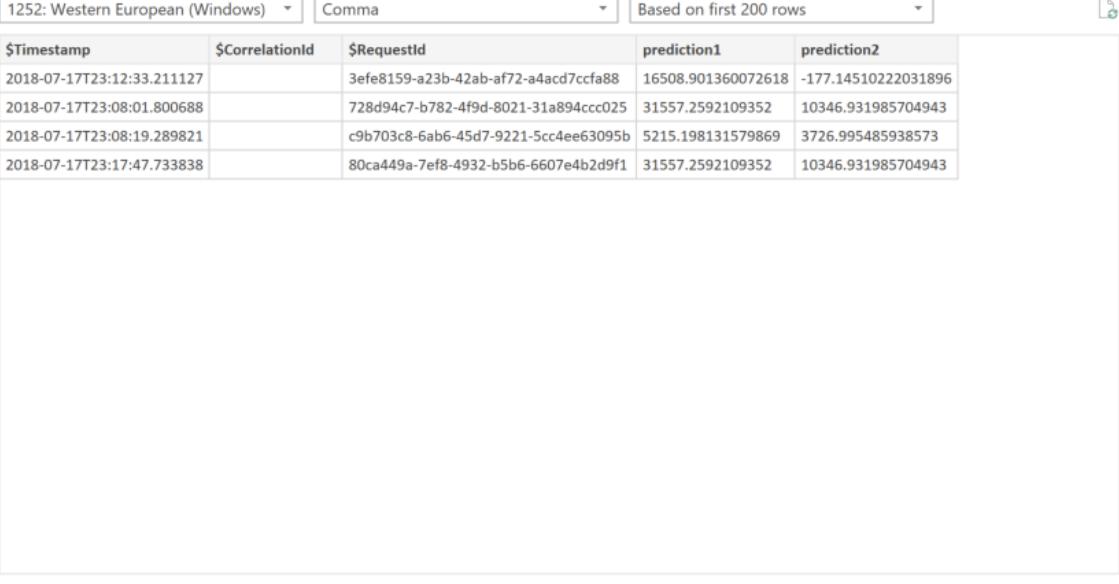
Specify the settings for each file. [Learn more](#)

Example File: First file

File Origin: 1252: Western European (Windows) Delimiter: Comma Data Type Detection: Based on first 200 rows

\$Timestamp	\$CorrelationId	\$RequestId	prediction1	prediction2
2018-07-17T23:12:33.211127		3efe8159-a23b-42ab-af72-a4acd7ccfa88	16508.901360072618	-177.14510222031896
2018-07-17T23:08:01.800688		728d94c7-b782-4f9d-8021-31a894ccc025	31557.2592109352	10346.931985704943
2018-07-17T23:08:19.289821		c9b703c8-6ab6-45d7-9221-5cc4ee63095b	5215.198131579869	3726.995485938573
2018-07-17T23:17:47.733838		80ca449a-7ef8-4932-b5b6-6607e4b2d9f1	31557.2592109352	10346.931985704943

Skip files with errors OK Cancel



9. You can now click **Close and Apply**.
10. If you added inputs and predictions your tables will automatically correlate by **RequestId**.
11. Start building your custom reports on your model data.

Analyzing model data using Databricks

1. Create a [Databricks workspace](#).
2. Go to your Databricks workspace.
3. In your databricks workspace select **Upload Data**.



Azure Databricks



Explore the Quickstart Tutorial

Spin up a cluster, run queries on preloaded data, and display results in 5 minutes.

Quickly im

Common Tasks

New Notebook

Upload Data

Create Table

New Cluster

New Job

Import Library

Read Documentation

Recents

2018-11-0

2018-06-1

setup2

2018-07-1

setup

4. Create New Table and select **Other Data Sources** -> Azure Blob Storage -> Create Table in Notebook.

Create New Table

Data source

Upload File DBFS Other Data Sources

Connector

Azure Blob Storage

Create Table in Notebook

5. Update the location of your data. Here is an example:

```
file_location = "wasbs://mycontainer@storageaccountname.blob.core.windows.net/modeldata/1a2b3c4d-5e6f-7g8h-9i10-j11k12l13m14/myresourcegrp/myWorkspace/aks-w-collv9/best_model/10/inputs/2018/*/*/data.csv"
file_type = "csv"
```

Cmd 2

Step 1: Set the data location and type

There are two ways to access Azure Blob storage: account keys and shared access signatures (SAS).

To get started, we need to set the location and type of the file.

Cmd 3

```
1 storage_account_name = "mystorage"
2 storage_account_access_key = "Jhsduhwe908rjjfoieudnf 98h v9udfhgb987yh g908ufgb98yfgb9 ufgb78gy8 gfo89ug98yfdg7yfg8ytg9fyg87"
```

Cmd 4

```
1 file_location = "wasbs://mycontainer@storageaccountname.blob.core.windows.net/modeldata/1a2b3c4d-5e6f-7g8h-9i10-j11k12l13m14/myresourcegrp/myWorkspace/aks-w-collv9/best_model/10/inputs/2018/*/*/data.csv"
2 file_type = "csv"
```

Cmd 5

```
1 spark.conf.set(
2   "fs.azure.account.key."+storage_account_name+".blob.core.windows.net",
3   storage_account_access_key)
```

6. Follow the steps on the template in order to view and analyze your data.

Example notebook

The [how-to-use-azureml/deployment/enable-data-collection-for-models-in-aks/enable-data-collection-for-models-in-aks.ipynb](#) notebook demonstrates concepts in this article.

Learn how to run notebooks by following the article, [Use Jupyter notebooks to explore this service](#).

Monitor your Azure Machine Learning models with Application Insights

12/11/2018 • 2 minutes to read • [Edit Online](#)

In this article, you learn how to set up Azure Application Insights for your Azure Machine Learning service. Application Insights gives you the opportunity to monitor:

- Request rates, response times, and failure rates.
- Dependency rates, response times, and failure rates.
- Exceptions.

[Learn more about Application Insights.](#)

Prerequisites

- If you don't have an Azure subscription, create a free account before you begin. Try the [free or paid version of Azure Machine Learning service](#) today.
- An Azure Machine Learning workspace, a local directory that contains your scripts, and the Azure Machine Learning SDK for Python installed. To learn how to get these prerequisites, see [How to configure a development environment](#).
- A trained machine learning model to be deployed to Azure Kubernetes Service (AKS) or Azure Container Instance (ACI). If you don't have one, see the [Train image classification model](#) tutorial.

Enable and disable from the SDK

Update a deployed service

1. Identify the service in your workspace. The value for `ws` is the name of your workspace.

```
from azureml.core.webservice import Webservice  
aks_service= Webservice(ws, "my-service-name")
```

2. Update your service and enable Application Insights.

```
aks_service.update(enable_app_insights=True)
```

Log custom traces in your service

If you want to log custom traces, follow the standard deployment process for AKS or ACI in the [How to deploy and where](#) document. Then use the following steps:

1. Update the scoring file by adding print statements.

```
print ("model initialized" + time.strftime("%H:%M:%S"))
```

2. Update the service configuration.

```
config = Webservice.deploy_configuration(enable_app_insights=True)
```

3. Build an image and deploy it on [AKS](#) or [ACI](#).

Disable tracking in Python

To disable Application Insights, use the following code:

```
## replace <service_name> with the name of the web service
<service_name>.update(enable_app_insights=False)
```

Enable and disable in the portal

You can enable and disable Application Insights in the Azure portal.

1. In the [Azure portal](#), open your workspace.
2. On the **Deployments** tab, select the service where you want to enable Application Insights.

The screenshot shows the Azure Machine Learning Workspace interface. At the top, there's a breadcrumb navigation: Home > All resources > Myworkspace > Myworkspace. Below that is the workspace title "Myworkspace" and subtitle "Machine Learning Workspace". A navigation bar has tabs for Experiments, Compute, Models, Images, Deployments (which is underlined), and Activities. The main area is titled "Deployments". It includes a toolbar with Refresh, Edit, and Delete buttons. A table lists five deployment entries:

NAME	LAST UPDATED
acimlc1	09/21/2018, 5:14:47 PM UTC
aks-w-dc2	09/19/2018, 8:29:22 PM UTC
aks-w-dc3	01/01/1, 12:00:00 AM UTC
aks-w-dc1	01/01/1, 12:00:00 AM UTC

3. Select **Edit**.

Home > All resources > Myworkspace > Myworkspace

Myworkspace

Machine Learning Workspace

Experiments Compute Models Images Deployments Activities

aks-w-dc2

[Back to Deployments](#) [Edit](#) [Delete](#)

Details Models Images

Attributes	
Description	
State	Healthy
Compute Type	AKS
Service ID	aks-w-dc2
Tags	
Creation date	09/19/2018, 8:04:06 PM UTC
Last updated	09/19/2018, 8:29:22 PM UTC

4. In **Advanced Settings**, select the **Enable AppInsights diagnostics** check box.

^ Advanced Settings

Enable AppInsights diagnostics

Enable Model data collection

Enable Authentication

5. Select **Update** at the bottom of the screen to apply the changes.

Disable

1. In the [Azure portal](#), open your workspace.
2. Select **Deployments**, select the service, and select **Edit**.

Home > All resources > Myworkspace > Myworkspace

Myworkspace

Machine Learning Workspace

Experiments Compute Models Images Deployments Activities

aks-w-dc2

[Back to Deployments](#) [Edit](#) [Delete](#)

Details Models Images

Attributes	
Description	
State	Healthy
Compute Type	AKS
Service ID	aks-w-dc2
Tags	
Creation date	09/19/2018, 8:04:06 PM UTC
Last updated	09/19/2018, 8:29:22 PM UTC

3. In **Advanced Settings**, clear the **Enable ApplInsights diagnostics** check box.

^ Advanced Settings

- Enable ApplInsights diagnostics
- Enable Model data collection
- Enable Authentication

4. Select **Update** at the bottom of the screen to apply the changes.

Evaluate data

Your service's data is stored in your Application Insights account, within the same resource group as your Azure Machine Learning service. To view it:

1. Go to your Machine Learning service workspace in the [Azure portal](#) and click on Application Insights link.

My Workspace
Machine Learning service workspace - PREVIEW

Search (Ctrl+ /) Delete

Overview Activity log Access control (IAM) Tags Diagnose and solve problems

Resource group <resource group>
Location East US 2
Subscription <subscription name>
Subscription ID <SubscriptionID>

Storage <Blob storage name>
Registry <registry name>
Key Vault <key vault name>
Application Insights <App Insights name>

2. Select the **Overview** tab to see a basic set of metrics for your service.

The screenshot shows the Azure Application Insights Analytics interface. On the left, there's a navigation pane with sections like Overview, Investigate, Metrics, and Usage. The main area displays three charts: 'Failed requests' (0), 'Server response time' (3ms), and 'Server requests' (10). Below the charts, there's a section for 'Availability'.

3. To look into your custom traces, select **Analytics**.

4. In the schema section, select **Traces**. Then select **Run** to run your query. Data should appear in a table format and should map to your custom calls in your scoring file.

The screenshot shows the Azure ML Studio Logs view. A Jupyter notebook cell is running, displaying Python code related to model deployment and monitoring. The output of the cell shows a table of trace logs. One specific row in the table is highlighted with a red box, showing a trace entry with a timestamp of 'Sep 19 2015 11:42:28' and a message indicating a prediction call to 'ridge'. The table has columns for timestamp, message, itemType, and customDimensions.

To learn more about how to use Application Insights, see [What is Application Insights?](#).

Example notebook

The [how-to-use-azureml/deployment/enable-app-insights-in-production-service/enable-app-insights-in-production-service.ipynb](#) notebook demonstrates concepts in this article.

Learn how to run notebooks by following the article, [Use Jupyter notebooks to explore this service](#).

Next steps

You can also collect data on your models in production. Read the article [Collect data for models in production](#).

Other references

- [Azure Monitor for containers](#)

Create and run a machine learning pipeline using Azure Machine Learning SDK

12/11/2018 • 6 minutes to read • [Edit Online](#)

In this article, you learn how to create, publish, run, and track a [machine learning pipeline](#) using the [Azure Machine Learning SDK](#). These pipelines help create and manage the workflows that stitch together various machine learning phases. Each phase of a pipeline, such as data preparation and model training, can include one or more steps.

The pipelines you create are visible to the members of your Azure Machine Learning service [workspace](#).

Pipelines use remote compute targets for computation and the storage of the intermediate and final data associated with that pipeline. Pipelines can read and write data to and from supported [Azure storage](#) locations.

NOTE

If you don't have an Azure subscription, create a free account before you begin. Try the [free or paid version of Azure Machine Learning service](#) today.

Prerequisites

- [Configure your development environment](#) to install the Azure Machine Learning SDK.
- Create an [Azure Machine Learning workspace](#) to hold all your pipeline resources.

```
ws = Workspace.create(  
    name = '<workspace-name>',  
    subscription_id = '<subscription-id>',  
    resource_group = '<resource-group>',  
    location = '<workspace_region>',  
    exist_ok = True)
```

Set up machine learning resources

Create the resources required to run a pipeline:

- Set up a datastore used to access the data needed in the pipeline steps.
- Configure a `DataReference` object to point to data that lives in or is accessible in a datastore.
- Set up the [compute targets](#) on which your pipeline steps will run.

Set up a datastore

A datastore stores the data for the pipeline to access. Each workspace has a default datastore. You can register additional datastores.

When you create your workspace, an [Azure file storage](#) and a [blob storage](#) are attached to the workspace by default. Azure file storage is the "default datastore" for a workspace, but you can also use blob storage as a datastore. Learn more about [Azure storage options](#).

```
# Default datastore (Azure file storage)
def_data_store = ws.get_default_datastore()

# The above call is equivalent to this
def_data_store = Datastore(ws, "workspacefilestore")

# Get blob storage associated with the workspace
def_blob_store = Datastore(ws, "workspaceblobstore")
```

Upload data files or directories to the datastore for them to be accessible from your pipelines. This example uses the blob storage version of the datastore:

```
def_blob_store.upload_files(
    ["./data/20news.pkl"],
    target_path="20newsgroups",
    overwrite=True)
```

A pipeline consists of one or more steps. A step is a unit run on a compute target. Steps might consume data sources and produce “intermediate” data. A step can create data such as a model, a directory with model and dependent files, or temporary data. This data is then available for other steps later in the pipeline.

Configure data reference

You just created a data source that can be referenced in a pipeline as an input to a step. A data source in a pipeline is represented by a [DataReference](#) object. The `DataReference` object points to data that lives in or is accessible from a datastore.

```
blob_input_data = DataReference(
    datastore=def_blob_store,
    data_reference_name="test_data",
    path_on_datastore="20newsgroups/20news.pkl")
```

Intermediate data (or output of a step) is represented by a [PipelineData](#) object. `output_data1` is produced as the output of a step and used as the input of one or more future steps. `PipelineData` introduces a data dependency between steps and creates an implicit execution order in the pipeline.

```
output_data1 = PipelineData(
    "output_data1",
    datastore=def_blob_store,
    output_name="output_data1")
```

Set up compute

In Azure Machine Learning, compute (or compute target) refers to the machines or clusters that will perform the computational steps in your machine learning pipeline. For example, you can create an Azure Machine Learning Compute for running your steps.

```

compute_name = "aml-compute"
if compute_name in ws.compute_targets:
    compute_target = ws.compute_targets[compute_name]
    if compute_target and type(compute_target) is AmlCompute:
        print('Found compute target: ' + compute_name)
else:
    print('Creating a new compute target...')
    provisioning_config = AmlCompute.provisioning_configuration(vm_size = vm_size, # NC6 is GPU-enabled
                                                               min_nodes = 1,
                                                               max_nodes = 4)
    # create the compute target
    compute_target = ComputeTarget.create(ws, compute_name, provisioning_config)

    # 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)

    # For a more detailed view of current cluster status, use the 'status' property
    print(compute_target.status.serialize())

```

Construct your pipeline steps

Now you are ready to define a pipeline step. There are many built-in steps available via the Azure Machine Learning SDK. The most basic of these steps is a `PythonScriptStep` that executes a Python script in a specified compute target.

```

trainStep = PythonScriptStep(
    script_name="train.py",
    arguments=["--input", blob_input_data, "--output", processed_data1],
    inputs=[blob_input_data],
    outputs=[processed_data1],
    compute_target=compute_target,
    source_directory=project_folder
)

```

After you define your steps, you build the pipeline using some or all of those steps.

NOTE

No file or data is uploaded to Azure Machine Learning service when you define the steps or build the pipeline.

```

# list of steps to run
compareModels = [trainStep, extractStep, compareStep]

# Build the pipeline
pipeline1 = Pipeline(workspace=ws, steps=[compareModels])

```

Submit the pipeline

When you submit the pipeline, the dependencies are checked for each step and a snapshot of the folder specified as the source directory is uploaded to Azure Machine Learning service. If no source directory is specified, the current local directory is uploaded.

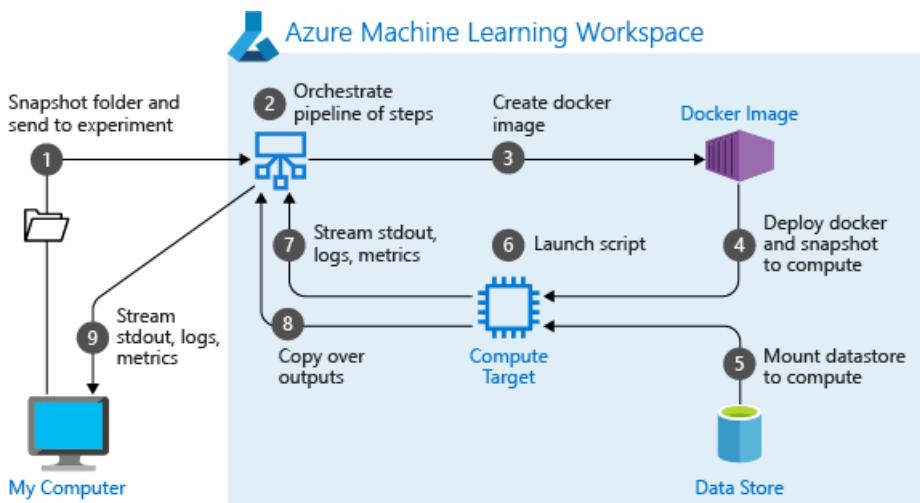
```

# Submit the pipeline to be run
pipeline_run1 = Experiment(ws, 'Compare_Models_Exp').submit(pipeline1)

```

When you first run a pipeline:

- The project snapshot is downloaded to the compute target from blob storage associated with the workspace.
- A docker image is built corresponding to each step in the pipeline.
- The docker image for each step is downloaded to the compute target from the container registry.
- If a `DataReference` object is specified in a step, the data store is mounted. If mount is not supported, the data is instead copied to the compute target.
- The step runs in the compute target specified in the step definition.
- Artifacts such as logs, stdout and stderr, metrics, and output specified by the step are created. These artifacts are then uploaded and kept in the user's default data store.



Publish a pipeline

You can publish a pipeline to run it with different inputs later. For the REST endpoint of an already published pipeline to accept parameters, the pipeline must be parameterized before publishing.

1. To create a pipeline parameter, use a `PipelineParameter` object with a default value.

```
pipeline_param = PipelineParameter(  
    name="pipeline_arg",  
    default_value=10)
```

2. Add this `PipelineParameter` object as a parameter to any of the steps in the pipeline as follows:

```
compareStep = PythonScriptStep(  
    script_name="compare.py",  
    arguments=["--comp_data1", comp_data1, "--comp_data2", comp_data2, "--output_data", out_data3, "--  
param1", pipeline_param],  
    inputs=[comp_data1, comp_data2],  
    outputs=[out_data3],  
    target=compute_target,  
    source_directory=project_folder)
```

3. Publish this pipeline that will accept a parameter when invoked.

```
published_pipeline1 = pipeline1.publish(  
    name="My_Published_Pipeline",  
    description="My Published Pipeline Description")
```

Run a published pipeline

All published pipelines have a REST endpoint to invoke the run of the pipeline from external systems such as non-Python clients. This endpoint provides a way for "managed repeatability" in batch scoring and retraining scenarios.

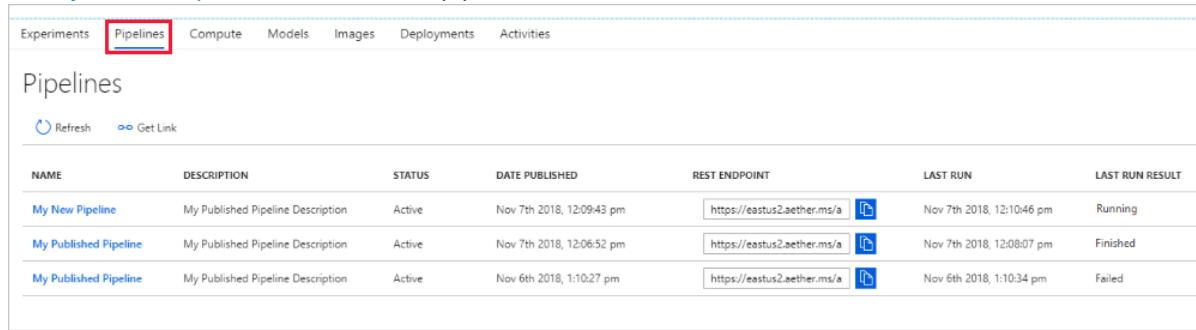
To invoke the run of the preceding pipeline, you need an Azure Active Directory authentication header token as described in [AzureCliAuthentication class](#)

```
response = requests.post(published_pipeline1.endpoint,
    headers=aad_token,
    json={"ExperimentName": "My_Pipeline",
        "ParameterAssignments": {"pipeline_arg": 20}})
```

View results

See the list of all your pipelines and their run details:

1. Sign in to the [Azure portal](#).
2. [View your workspace](#) to find the list of pipelines.



NAME	DESCRIPTION	STATUS	DATE PUBLISHED	REST ENDPOINT	LAST RUN	LAST RUN RESULT
My New Pipeline	My Published Pipeline Description	Active	Nov 7th 2018, 12:09:43 pm	https://eastus2.aether.ms/a 	Nov 7th 2018, 12:10:46 pm	Running
My Published Pipeline	My Published Pipeline Description	Active	Nov 7th 2018, 12:06:52 pm	https://eastus2.aether.ms/a 	Nov 7th 2018, 12:08:07 pm	Finished
My Published Pipeline	My Published Pipeline Description	Active	Nov 6th 2018, 1:10:27 pm	https://eastus2.aether.ms/a 	Nov 6th 2018, 1:10:34 pm	Failed

3. Select a specific pipeline to see the run results.

Next steps

- Use [these Jupyter notebooks on GitHub](#) to explore machine learning pipelines further.
- Read the SDK reference help for the [azureml-pipelines-core](#) package and the [azureml-pipelines-steps](#) package.

Learn how to run notebooks by following the article, [Use Jupyter notebooks to explore this service](#).

Manage and request quotas for Azure resources

12/7/2018 • 7 minutes to read • [Edit Online](#)

As with other Azure services, there are limits on certain resources associated with the Azure Machine Learning service. These limits range from a cap on the number of workspaces you can create to limits on the actual underlying compute that gets used for training or inferencing your models. This article gives more details on the pre-configured limits on various Azure resources for your subscription and also contains handy links to request quota enhancements for each type of resource. These limits are put in place to prevent budget over-runs due to fraud, and to honor Azure capacity constraints.

Keep these quotas in mind as you design and scale up your Azure ML resources for production workloads. For example, if your cluster doesn't reach the target number of nodes you specified, then you might have reached a Azure Machine Learning Compute cores limit for your subscription. If you want to raise the limit or quota above the Default Limit, open an online customer support request at no charge. The limits can't be raised above the Maximum Limit value shown in the following tables due to Azure Capacity constraints. If there is no Maximum Limit column, then the resource doesn't have adjustable limits.

Special considerations

- A quota is a credit limit, not a capacity guarantee. If you have large-scale capacity needs, contact Azure support.
- Your quota is shared across all the services in your subscriptions including Azure Machine Learning service. The only exception is Azure Machine Learning compute which has a separate quota from the core compute quota. Be sure to calculate the quota usage across all services when evaluating your capacity needs.
- Default limits vary by offer Category Type, such as Free Trial, Pay-As-You-Go, and series, such as Dv2, F, G, and so on.

Default resource quotas

Here is a breakdown of the quota limits by various resource types within your Azure subscription.

IMPORTANT

Limits are subject to change. The latest can always be found at the service-level quota [document](#) for all of Azure.

Virtual machines

There is a limit on the number of virtual machines you can provision on an Azure subscription across your services or in a standalone manner. This limit is at the region level both on the total cores and also on a per family basis.

It is important to emphasize that virtual machine cores have a regional total limit and a regional per size series (Dv2, F, etc.) limit that are separately enforced. For example, consider a subscription with a US East total VM core limit of 30, an A series core limit of 30, and a D series core limit of 30. This subscription would be allowed to deploy 30 A1 VMs, or 30 D1 VMs, or a combination of the two not to exceed a total of 30 cores (for example, 10 A1 VMs and 20 D1 VMs).

RESOURCE	DEFAULT LIMIT	MAXIMUM LIMIT
VMs per subscription	10,000 ¹ per Region	10,000 per Region
VM total cores per subscription	20 ¹ per Region	Contact support
VM per series (Dv2, F, etc.) cores per subscription	20 ¹ per Region	Contact support
Co-administrators per subscription	Unlimited	Unlimited
Storage accounts per region per subscription	200	200 ²
Resource Groups per subscription	980	980
Availability Sets per subscription	2,000 per Region	2,000 per Region
Resource Manager API request size	4,194,304 bytes	4,194,304 bytes
Tags per subscription ³	unlimited	unlimited
Unique tag calculations per subscription ³	10,000	10,000
Cloud services per subscription	Not Applicable ⁴	Not Applicable ⁴
Affinity groups per subscription	Not Applicable ⁴	Not Applicable ⁴
Subscription level deployments per location	800	800

¹Default limits vary by offer Category Type, such as Free Trial, Pay-As-You-Go, and series, such as Dv2, F, G, etc.

²This includes both Standard and Premium storage accounts. If you require more than 200 storage accounts, make a request through [Azure Support](#). The Azure Storage team will review your business case and may approve up to 250 storage accounts.

³You can apply an unlimited number of tags per subscription. The number of tags per resource or resource group is limited to 15. Resource Manager only returns a [list of unique tag name and values](#) in the subscription when the number of tags is 10,000 or less. However, you can still find a resource by tag when the number exceeds 10,000.

⁴These features are no longer required with Azure Resource Groups and the Azure Resource Manager.

NOTE

It is important to emphasize that virtual machine cores have a regional total limit as well as a regional per size series (Dv2, F, etc.) limit that are separately enforced. For example, consider a subscription with a US East total VM core limit of 30, an A series core limit of 30, and a D series core limit of 30. This subscription would be allowed to deploy 30 A1 VMs, or 30 D1 VMs, or a combination of the two not to exceed a total of 30 cores (for example, 10 A1 VMs and 20 D1 VMs).

For a more detailed and up-to-date list of quota limits, check the Azure-wide quota article [here](#).

Azure Machine Learning Compute

For Azure Machine Learning Compute, there is a default quota limit on both the number of cores and number of

unique compute resources allowed per region in a subscription. This quota is separate from the VM core quota above and the core limits are not shared currently between the two resource types.

Available resources:

- Dedicated cores per region have a default limit of 10 - 24. The number of dedicated cores per subscription can be increased. Contact Azure support to discuss increase options.
- Low-priority cores per region have a default limit of 10 - 24. The number of low-priority cores per subscription can be increased. Contact Azure support to discuss increase options.
- Clusters per region have a default limit of 100 and a maximum limit of 200. Contact Azure support if you want to request an increase beyond this limit.
- There are **other strict limits** which cannot be exceeded once hit.

RESOURCE	MAXIMUM LIMIT
Maximum workspaces per resource group	800
Maximum nodes in a single Azure Machine Learning Compute (AmlCompute) resource	100 nodes
Maximum GPU MPI processes per node	1-4
Maximum GPU workers per node	1-4
Maximum job lifetime	7 days ¹
Maximum parameter servers per node	1

¹ The maximum lifetime refers to the time that a run starts and when it finishes. Completed runs persist indefinitely; data for runs not completed within the maximum lifetime is not accessible.

Container instances

There is also a limit on the number of container instances that you can spin up in a given time period (scoped hourly) or across your entire subscription.

RESOURCE	DEFAULT LIMIT
Container groups per subscription	100 ¹
Number of containers per container group	60
Number of volumes per container group	20
Ports per IP	5
Container creates per hour	300 ¹
Container creates per 5 minutes	100 ¹
Container deletes per hour	300 ¹
Container deletes per 5 minutes	100 ¹

RESOURCE	DEFAULT LIMIT
Multiple containers per container group	Linux only ²
Azure Files volumes	Linux only ²
GitRepo volumes	Linux only ²
Secret volumes	Linux only ²

¹ Create an [Azure support request](#) to request a limit increase.

² Windows support for this feature is planned.

For a more detailed and up-to-date list of quota limits, check the Azure-wide quota article [here](#).

Storage

There is a limit on the number of storage accounts per region as well in a given subscription. The default limit is 200 and includes both Standard and Premium Storage accounts. If you require more than 200 storage accounts in a given region, make a request through [Azure Support](#). The Azure Storage team will review your business case and may approve up to 250 storage accounts for a given region.

Find your quotas

Viewing your quota for various resources, such as Virtual Machines, Storage, Network, is easy through the Azure portal.

1. On the left pane, select **All services** and then select **Subscriptions** under the General category.
2. From the list of subscriptions, select the subscription whose quota you are looking for.
3. On the left pane, select **Machine Learning service** and then select any workspace from the list shown
4. On the next blade, under the **Support + troubleshooting section** select **Usage + quotas** to view your current quota limits and usage.
5. Select a subscription to view the quota limits. Remember to filter to the region you are interested in.

Request quota increases

If you want to raise the limit or quota above the default limit, [open an online customer support request](#) at no charge.

The limits can't be raised above the maximum limit value shown in the tables. If there is no maximum limit, then the resource doesn't have adjustable limits. [This](#) article covers the quota increase process in more detail.

When requesting a quota increase, you need to select the service you are requesting to raise the quota against, which could be services such as Machine Learning service quota, Container instances or Storage quota. In addition for Azure Machine Learning Compute, you can simply click on the **Request Quota** button while viewing the quota following the steps above.

NOTE

Free Trial subscriptions are not eligible for limit or quota increases. If you have a [Free Trial subscription](#), you can upgrade to a Pay-As-You-Go subscription. For more information, see [Upgrade Azure Free Trial to Pay-As-You-Go](#) and [Free Trial subscription FAQ](#).

Use the CLI extension for Azure Machine Learning service

12/11/2018 • 2 minutes to read • [Edit Online](#)

The Azure Machine Learning CLI is an extension to the [Azure CLI](#), a cross-platform command-line interface for the Azure platform. This extension provides commands for working with the Azure Machine Learning service from the command line. It allows you to create scripts that automate your machine learning workflows. For example, you can create scripts that perform the following actions:

- Run experiments to create machine learning models
- Register machine learning models for customer usage
- Package, deploy, and track the lifecycle of your machine learning models

The CLI is not a replacement for the Azure Machine Learning SDK. It is a complementary tool that is optimized to handle highly parameterized tasks such as:

- Creating compute resources
- Parameterized experiment submission
- Model registration
- Image creation
- Service deployment

Prerequisites

- To use the CLI, you must have an Azure subscription. If you don't have an Azure subscription, create a free account before you begin. Try the [free or paid version of Azure Machine Learning service](#) today.
- The [Azure CLI](#).

Install the extension

To install the Machine Learning CLI extension, use the following command:

```
az extension add -s https://azuremlsdktestpypi.blob.core.windows.net/wheels/sdk-release/Preview/E7501C02541B433786111FE8E140CAA1/azure_cli_ml-1.0.2-py2.py3-none-any.whl --pip-extra-index-urls https://azuremlsdktestpypi.azureedge.net/sdk-release/Preview/E7501C02541B433786111FE8E140CAA1
```

When prompted, select `y` to install the extension.

To verify that the extension has been installed, use the following command to display a list of ML-specific subcommands:

```
az ml -h
```

TIP

To update the extension you must **remove** it, and then **install** it. This installs the latest version.

Remove the extension

To remove the CLI extension, use the following command:

```
az extension remove -n azure-cli-ml
```

Resource management

The following commands demonstrate how to use the CLI to manage resources used by Azure Machine Learning.

- Create an Azure Machine Learning service workspace:

```
az ml workspace create -n myworkspace -g myresourcegroup
```

- Set a default workspace:

```
az configure --defaults aml_workspace=myworkspace group=myresourcegroup
```

- Create a managed compute target for distributed training:

```
az ml computetarget create amlcompute -n mycompute --max_nodes 4 --size Standard_NC6
```

- Update a managed compute target:

```
az ml computetarget update --name mycompute --workspace --group --max_nodes 4 --min_nodes 2 --idle_time 300
```

- Attach an unmanaged compute target for training or deployment:

```
az ml computetarget attach aks -n myaks -i myaksresourceid -g myrg -w myworkspace
```

Experiments

The following commands demonstrate how to use the CLI to work with experiments:

- Attach a project (run configuration) before submitting an experiment:

```
az ml project attach --experiment-name myhistory
```

- Start a run of your experiment. When using this command, specify a compute target. In this example, `local` uses the local computer to train the model using the `train.py` script:

```
az ml run submit -c local train.py
```

- View a list of submitted experiments:

```
az ml history list
```

Model registration, image creation & deployment

The following commands demonstrate how to register a trained model, and then deploy it as a production service:

- Register a model with Azure Machine Learning:

```
az ml model register -n mymodel -m sklearn_regression_model.pkl
```

- Create an image that contains your machine learning model and dependencies:

```
az ml image create container -n myimage -r python -m mymodel:1 -f score.py -c myenv.yml
```

- Deploy an image to a compute target:

```
az ml service create aci -n myaciservice --image-id myimage:1
```

Known issues and troubleshooting Azure Machine Learning service

12/10/2018 • 2 minutes to read • [Edit Online](#)

This article helps you find and correct errors or failures encountered when using the Azure Machine Learning service.

SDK installation issues

Error message: Cannot uninstall 'PyYAML'

Azure Machine Learning SDK for Python: PyYAML is a distutils installed project. Therefore, we cannot accurately determine which files belong to it in the event of a partial uninstall. To continue installing the SDK while ignoring this error, use:

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

Trouble creating Azure Machine Learning Compute

There is a rare chance that some users who created their Azure Machine Learning workspace from the Azure portal before the GA release might not be able to create Azure Machine Learning Compute in that workspace. You can either raise a support request against the service or create a new workspace through the Portal or the SDK to unblock yourself immediately.

Image building failure

Image building failure when deploying web service. Workaround is to add "pynacl==1.2.1" as a pip dependency to Conda file for image configuration.

FPGAs

You will not be able to deploy models on FPGAs until you have requested and been approved for FPGA quota. To request access, fill out the quota request form: <https://aka.ms/aml-real-time-ai>

Databricks

Databricks and Azure Machine Learning issues.

1. Databricks cluster recommendation:

Create your Azure Databricks cluster as v4.x with Python 3. We recommend a high concurrency cluster.

2. AML SDK install failure on Databricks when more packages are installed.

Some packages, such as `psutil`, can cause conflicts. To avoid installation errors, install packages by freezing lib version. This issue is related to Databricks and not related to Azure ML SDK - you may face it with other libs too. Example:

```
psutil cryptography==1.5 pyopenssl==16.0.0 ipython==2.2.0
```

Alternatively, you can use init scripts if you keep facing install issues with Python libs. This approach is not an officially supported approach. You can refer to [this doc](#).

3. When using Automated Machine Learning on Databricks, if you see

```
Import error: numpy.core.multiarray failed to import
```

Workaround: import Python library `numpy==1.14.5` to your Databricks cluster using Create a library to [install and attach](#).

Azure portal

If you go directly to view your workspace from a share link from the SDK or the portal, you will not be able to view the normal Overview page with subscription information in the extension. You will also not be able to switch into another workspace. If you need to view another workspace, the workaround is to go directly to the [Azure portal](#) and search for the workspace name.

Diagnostic logs

Sometimes it can be helpful if you can provide diagnostic information when asking for help. Here is where the log files live:

Resource quotas

Learn about the [resource quotas](#) you might encounter when working with Azure Machine Learning.

Get more support

You can submit requests for support and get help from technical support, forums, and more. [Learn more...](#)

Get support and training for Azure Machine Learning service

12/10/2018 • 2 minutes to read • [Edit Online](#)

This article provides information on how to learn more about Azure Machine Learning service as well as get support for your issues and questions.

Learn more about Azure Machine Learning

In addition to the documentation on this site, you can find:

- [Tutorials and how-to articles](#)
- [Architecture overview](#)
- [Videos](#)

Submit doc feedback

You can **submit requests** for additional learning materials using the feedback link at the bottom of the article.

Get support for Azure Machine Learning service

Check out these support resources:

- **Technical support:** Visit [Azure technical support](#) and select Machine Learning.
- **User forum:** Ask questions, answer questions, and connect with other users in the [Azure Machine Learning service support forum on MSDN](#).
- **Stack Overflow:** Visit the Azure Machine Learning community on [StackOverflow](#) tagged with "Azure-Machine-Learning".
- **Share product suggestions** and feature requests in our [Azure Machine Learning Feedback Channel](#), which is also accessible using the link at the bottom of each article.

Export or delete your Machine Learning service workspace data

12/10/2018 • 3 minutes to read • [Edit Online](#)

In Azure Machine Learning, you can export or delete your workspace data with the authenticated REST API. This article tells you how.

NOTE

If you're interested in viewing or deleting personal data, please see the [Azure Data Subject Requests for the GDPR](#) article. If you're looking for general info about GDPR, see the [GDPR section of the Service Trust portal](#).

NOTE

This article provides steps for how to delete personal data from the device or service and can be used to support your obligations under the GDPR. If you're looking for general info about GDPR, see the [GDPR section of the Service Trust portal](#).

Control your workspace data

In-product data stored by Azure Machine Learning Services is available for export and deletion through the Azure portal, CLI, SDK, and authenticated REST APIs. Telemetry data can be accessed through the Azure Privacy portal.

In Azure Machine Learning Services, personal data consists of user information in run history documents and telemetry records of some user interactions with the service.

Delete workspace data with the REST API

In order to delete data, the following API calls can be made with the HTTP DELETE verb. These are authorized by having an `Authorization: Bearer <arm-token>` header in the request, where `<arm-token>` is the AAD access token for the `https://management.core.windows.net/` endpoint.

To learn how to get this token and call Azure endpoints, see [Azure REST API documentation](#).

In the examples following, replace the text in {} with the instance names that determine the associated resource.

Delete an entire workspace

Use this call to delete an entire workspace.

WARNING

All information will be deleted and the workspace will no longer be usable.

```
https://management.azure.com/subscriptions/{subscriptionId}/resourceGroups/{resourceGroupName}/providers/Microsoft.MachineLearningServices/workspaces/{workspaceName}?api-version=2018-03-01-preview
```

Delete models

Use this call to get a list of models and their IDs:

```
https://{{location}}.modelmanagement.azureml.net/api/subscriptions/{{subscriptionId}}/resourceGroups/{{resourceGroupName}}/providers/Microsoft.MachineLearningServices/workspaces/{{workspace}}/models?api-version=2018-03-01-preview
```

Individual models can be deleted with:

```
https://{{location}}.modelmanagement.azureml.net/api/subscriptions/{{subscriptionId}}/resourceGroups/{{resourceGroupName}}/providers/Microsoft.MachineLearningServices/workspaces/{{workspace}}/models/{{id}}?api-version=2018-03-01-preview
```

Delete assets

Use this call to get a list of assets and their IDs:

```
https://{{location}}.modelmanagement.azureml.net/api/subscriptions/{{subscriptionId}}/resourceGroups/{{resourceGroupName}}/providers/Microsoft.MachineLearningServices/workspaces/{{workspace}}/assets?api-version=2018-03-01-preview
```

Individual assets can be deleted with:

```
https://{{location}}.modelmanagement.azureml.net/api/subscriptions/{{subscriptionId}}/resourceGroups/{{resourceGroupName}}/providers/Microsoft.MachineLearningServices/workspaces/{{workspace}}/assets/{{id}}?api-version=2018-03-01-preview
```

Delete images

Use this call to get a list of images and their IDs:

```
https://{{location}}.modelmanagement.azureml.net/api/subscriptions/{{subscriptionId}}/resourceGroups/{{resourceGroupName}}/providers/Microsoft.MachineLearningServices/workspaces/{{workspace}}/images?api-version=2018-03-01-preview
```

Individual images can be deleted with:

```
https://{{location}}.modelmanagement.azureml.net/api/subscriptions/{{subscriptionId}}/resourceGroups/{{resourceGroupName}}/providers/Microsoft.MachineLearningServices/workspaces/{{workspace}}/images/{{id}}?api-version=2018-03-01-preview
```

Delete services

Use this call to get a list of services and their IDs:

```
https://{{location}}.modelmanagement.azureml.net/api/subscriptions/{{subscriptionId}}/resourceGroups/{{resourceGroupName}}/providers/Microsoft.MachineLearningServices/workspaces/{{workspace}}/services?api-version=2018-03-01-preview
```

Individual services can be deleted with:

```
https://{{location}}.modelmanagement.azureml.net/api/subscriptions/{{subscriptionId}}/resourceGroups/{{resourceGroupName}}/providers/Microsoft.MachineLearningServices/workspaces/{{workspace}}/services/{{id}}?api-version=2018-03-01-preview
```

Export service data with the REST API

In order to export data, the following API calls can be made with the HTTP GET verb. These are authorized by having an `Authorization: Bearer <arm-token>` header in the request, where `<arm-token>` is the AAD access token

for the endpoint `https://management.core.windows.net/`

To learn how to get this token and call Azure endpoints, see [Azure REST API documentation](#).

In the examples following, replace the text in {} with the instance names that determine the associated resource.

Export Workspace information

Use this call to get a list of all workspaces:

```
https://management.azure.com/subscriptions/{subscriptionId}/resourceGroups/{resourceGroupName}/providers/Microsoft.MachineLearningServices/workspaces?api-version=2018-03-01-preview
```

Information about an individual workspace can be obtained by:

```
https://management.azure.com/subscriptions/{subscriptionId}/resourceGroups/{resourceGroupName}/providers/Microsoft.MachineLearningServices/workspaces/{workspaceName}?api-version=2018-03-01-preview
```

Export Compute Information

All compute targets attached to a workspace can be obtained by:

```
https://management.azure.com/subscriptions/{subscriptionId}/resourceGroup/{resourceGroupName}/providers/Microsoft.MachineLearningServices/workspaces/{workspaceName}/computes?api-version=2018-03-01-preview
```

Information about a single compute target can be obtained by:

```
https://management.azure.com/subscriptions/{subscriptionId}/resourceGroup/{resourceGroupName}/providers/Microsoft.MachineLearningServices/workspaces/{workspaceName}/computes/{computeName}?api-version=2018-03-01-preview
```

Export run history data

Use this call to get a list of all experiments and their information:

```
https://{{location}}.experiments.azureml.net/history/v1.0/subscriptions/{subscriptionId}/resourceGroups/{resourceGroupName}/providers/Microsoft.MachineLearningServices/workspaces/{workspaceName}/experiments
```

All the runs for a particular experiment can be obtained by:

```
https://{{location}}.experiments.azureml.net/history/v1.0/subscriptions/{subscriptionId}/resourceGroups/{resourceGroupName}/providers/Microsoft.MachineLearningServices/workspaces/{workspaceName}/experiments/{experimentName}/runs
```

Run history items can be obtained by:

```
https://{{location}}.experiments.azureml.net/history/v1.0/subscriptions/{subscriptionId}/resourceGroups/{resourceGroupName}/providers/Microsoft.MachineLearningServices/workspaces/{workspaceName}/experiments/{experimentName}/runs/{runId}
```

All run metrics for an experiment can be obtained by:

```
https://{{location}}.experiments.azureml.net/history/v1.0/subscriptions/{subscriptionId}/resourceGroups/{resourceGroupName}/providers/Microsoft.MachineLearningServices/workspaces/{workspaceName}/experiments/{experimentName}/metrics
```

A single run metric can be obtained by:

```
https://{{location}}.experiments.azureml.net/history/v1.0/subscriptions/{{subscriptionId}}/resourceGroups/{{resourceGroupName}}/providers/Microsoft.MachineLearningServices/workspaces/{{workspaceName}}/experiments/{{experimentName}}/metrics/{{metricId}}
```

Export artifacts

Use this call to get a list of artifacts and their paths:

```
https://{{location}}.experiments.azureml.net/artifact/v1.0/subscriptions/{{subscriptionId}}/resourceGroups/{{resourceGroupName}}/providers/Microsoft.MachineLearningServices/workspaces/{{workspaceName}}/artifacts/origins/ExperimentRun/containers/{{runId}}
```

Export notifications

Use this call to get a list of stored tasks:

```
https://{{location}}.experiments.azureml.net/notification/v1.0/subscriptions/{{subscriptionId}}/resourceGroups/{{resourceGroupName}}/providers/Microsoft.MachineLearningServices/workspaces/{{workspaceName}}/tasks
```

Notifications for a single task can be obtained by:

```
https://{{location}}.experiments.azureml.net/notification/v1.0/subscriptions/{{subscriptionId}}/resourceGroups/{{resourceGroupName}}/providers/Microsoft.MachineLearningServices/workspaces/{{workspaceName}}/tasks/{{taskId}}
```

Export data stores

Use this call to get a list of data stores:

```
https://{{location}}.experiments.azureml.net/datastore/v1.0/subscriptions/{{subscriptionId}}/resourceGroups/{{resourceGroupName}}/providers/Microsoft.MachineLearningServices/workspaces/{{workspaceName}}/datastores
```

Individual data stores can be obtained by:

```
https://{{location}}.experiments.azureml.net/datastore/v1.0/subscriptions/{{subscriptionId}}/resourceGroups/{{resourceGroupName}}/providers/Microsoft.MachineLearningServices/workspaces/{{workspaceName}}/datastores/{{name}}
```

Export models

Use this call to get a list of models and their IDs:

```
https://{{location}}.modelmanagement.azureml.net/api/subscriptions/{{subscriptionId}}/resourceGroups/{{resourceGroupName}}/providers/Microsoft.MachineLearningServices/workspaces/{{workspace}}/models?api-version=2018-03-01-preview
```

Individual models can be obtained by:

```
https://{{location}}.modelmanagement.azureml.net/api/subscriptions/{{subscriptionId}}/resourceGroups/{{resourceGroupName}}/providers/Microsoft.MachineLearningServices/workspaces/{{workspace}}/models/{{id}}?api-version=2018-03-01-preview
```

Export assets

Use this call to get a list of assets and their IDs:

```
https://{{location}}.modelmanagement.azureml.net/api/subscriptions/{{subscriptionId}}/resourceGroups/{{resourceGroupName}}/providers/Microsoft.MachineLearningServices/workspaces/{{workspace}}/assets?api-version=2018-03-01-preview
```

Individual assets can be obtained by:

```
https://{{location}}.modelmanagement.azureml.net/api/subscriptions/{{subscriptionId}}/resourceGroups/{{resourceGroupName}}/providers/Microsoft.MachineLearningServices/workspaces/{{workspace}}/assets/{{id}}?api-version=2018-03-01-preview
```

Export images

Use this call to get a list of images and their IDs:

```
https://{{location}}.modelmanagement.azureml.net/api/subscriptions/{{subscriptionId}}/resourceGroups/{{resourceGroupName}}/providers/Microsoft.MachineLearningServices/workspaces/{{workspace}}/images?api-version=2018-03-01-preview
```

Individual images can be obtained by:

```
https://{{location}}.modelmanagement.azureml.net/api/subscriptions/{{subscriptionId}}/resourceGroups/{{resourceGroupName}}/providers/Microsoft.MachineLearningServices/workspaces/{{workspace}}/images/{{id}}?api-version=2018-03-01-preview
```

Export services

Use this call to get a list of services and their IDs:

```
https://{{location}}.modelmanagement.azureml.net/api/subscriptions/{{subscriptionId}}/resourceGroups/{{resourceGroupName}}/providers/Microsoft.MachineLearningServices/workspaces/{{workspace}}/services?api-version=2018-03-01-preview
```

Individual services can be obtained by:

```
https://{{location}}.modelmanagement.azureml.net/api/subscriptions/{{subscriptionId}}/resourceGroups/{{resourceGroupName}}/providers/Microsoft.MachineLearningServices/workspaces/{{workspace}}/services/{{id}}?api-version=2018-03-01-preview
```

Export Pipeline Experiments

Individual experiments can be obtained by:

```
https://{{location}}.aether.ms/api/v1.0/subscriptions/{{subscriptionId}}/resourceGroups/{{resourceGroupName}}/providers/Microsoft.MachineLearningServices/workspaces/{{workspaceName}}/Experiments/{{experimentId}}
```

Export Pipeline Graphs

Individual graphs can be obtained by:

```
https://{{location}}.aether.ms/api/v1.0/subscriptions/{{subscriptionId}}/resourceGroups/{{resourceGroupName}}/providers/Microsoft.MachineLearningServices/workspaces/{{workspaceName}}/Graphs/{{graphId}}
```

Export Pipeline Modules

Modules can be obtained by:

```
https://{{location}}.aether.ms/api/v1.0/subscriptions/{{subscriptionId}}/resourceGroups/{{resourceGroupName}}/provide  
rs/Microsoft.MachineLearningServices/workspaces/{{workspaceName}}/Modules/{{id}}
```

Export Pipeline Templates

Templates can be obtained by:

```
https://{{location}}.aether.ms/api/v1.0/subscriptions/{{subscriptionId}}/resourceGroups/{{resourceGroupName}}/provide  
rs/Microsoft.MachineLearningServices/workspaces/{{workspaceName}}/Templates/{{templateId}}
```

Export Pipeline Data Sources

Data Sources can be obtained by:

```
https://{{location}}.aether.ms/api/v1.0/subscriptions/{{subscriptionId}}/resourceGroups/{{resourceGroupName}}/provide  
rs/Microsoft.MachineLearningServices/workspaces/{{workspaceName}}/DataSources/{{id}}
```

Migrate from Workbench to the latest version of Azure Machine Learning service

12/10/2018 • 4 minutes to read • [Edit Online](#)

If you have installed the Workbench application and/or have experimentation and model management preview accounts, use this article to migrate to the latest version. If you don't have preview Workbench installed, or an experimentation and/or model management account, you don't need to migrate anything.

What can I migrate?

Most artifacts created in the first preview of Azure Machine Learning service are stored in your own local or cloud storage. These artifacts won't disappear. To migrate, register the artifacts again with the updated Azure Machine Learning service.

The following table and article explain what you can do with your existing assets and resources before or after moving over to the latest version of Azure Machine Learning service. You can also continue to use the previous version and your assets for some time ([see transition support timeline](#)).

ASSET OR RESOURCE FROM OLD VERSION	CAN I MIGRATE?	ACTIONS
Machine learning models (as local files)	Yes	None. Works as before.
Model dependencies & schemas (as local files)	Yes	None. Works as before.
Projects	Yes	Attach the local folder to new workspace.
Run histories	No	Downloadable for a while.
Data prep files	No	Prepare any size data set for modeling using the new Azure Machine Learning Data Prep SDK or use Azure Databricks.
Compute targets	No	Register them in new workspace.
Registered models	No	Re-register the model under a new workspace.
Registered manifests	No	None. Manifests no longer exists as a concept in the new workspace.
Registered images	No	Re-create the deployment Docker image under a new workspace.
Deployed web services	No	None. They'll still work as-is or deploy them again using latest version .

ASSET OR RESOURCE FROM OLD VERSION	CAN I MIGRATE?	ACTIONS
Experimentation and Model management accounts	No	Create a workspace instead.
Machine learning CLI & SDK	No	Use the new CLI and SDK for new work.

Learn more about [what changed in this release?](#)

WARNING

This article is not for Azure Machine Learning Studio users. It is for Azure Machine Learning service customers who have installed the Workbench (preview) application and/or have experimentation and model management preview accounts.

Azure resources

Resources such as your experimentation accounts, model management accounts, and machine learning compute environments cannot be migrated over to the latest version of Azure Machine Learning service. See the [timeline](#) on how long your assets will continue to work.

Get started with the latest version by creating an Azure Machine Learning service workspace in the [Azure portal](#). The portal's workspace dashboard is supported on Edge, Chrome and Firefox browsers only.

This new workspace is the top-level service resource and enables you to use all of the latest features of Azure Machine Learning service. Learn more about this [workspace and architecture](#).

Projects

Instead of having your projects in a workspace in the cloud, projects are now directories on your local machine in the latest release. See a diagram of the [latest architecture](#).

To continue using the local directory containing your files and scripts, specify the directory's name in the '`experiment.submit`' Python command or using the 'az ml project attach' CLI command.

For example:

```
run = exp.submit(source_directory = script_folder, script = 'train.py', run_config =
run_config_system_managed)
```

Deployed web services

To migrate web services, redeploy your models using the new SDK or CLI to the new deployment targets. There is no need to change your original scoring file, model file dependencies files, environment file, and schema files.

In the latest version, models are deployed as web services to Azure Container Instances (ACI) or Azure Kubernetes Service (AKS) clusters.

Learn more in these articles:

- [How to deploy and where](#)
- [Tutorial: Deploy models with Azure Machine Learning service](#)

When [support for the previous CLI ends](#), you won't be able to manage the web services you originally deployed with your Model Management account. However, those web services will continue to work for as long as Azure Container Service (ACS) is still supported.

Run history records

While you can't continue to add to your existing run histories under the old workspace, you can export the histories you have using the previous CLI. When [support for the previous CLI ends](#), you won't be able to export these run histories anymore.

Start training your models and tracking the run histories using the new CLI and SDK. You can learn how with the [Tutorial: train models with Azure Machine Learning service](#).

To export the run history with previous CLI:

```
#list all runs  
az ml history list  
  
#get details about a particular run  
az ml history info  
  
# download all artifacts of a run  
az ml history download
```

Data preparation files

Data preparation files are not portable without the Workbench. But you can still prepare any size data set for modeling using the new Azure Machine Learning Data Prep SDK or use Azure Databricks for big data sets. [Learn how to get the data prep SDK](#).

Next steps

For a quickstart showing you how to create a workspace, create a project, run a script, and explore the run history of the script with the latest version of Azure Machine Learning service, try [get started with Azure Machine Learning service](#).

For a more in-depth experience of this workflow, follow the full-length tutorial that contains detailed steps for training and deploying models with Azure Machine Learning service.

[Tutorial: Train and deploy models](#)

Export or delete your experimentation or model management data in Machine Learning

9/25/2018 • 4 minutes to read • [Edit Online](#)

NOTE

This article is deprecated. Support for earlier versions of this service will end incrementally. [View the support timeline](#). Read the [newest article](#) on this topic.

In Azure Machine Learning, you can export or delete your account data related to experimentation or model management with the authenticated REST API. This article tells you how.

NOTE

If you're interested in viewing or deleting personal data, please see the [Azure Data Subject Requests for the GDPR](#) article. If you're looking for general info about GDPR, see the [GDPR section of the Service Trust portal](#).

NOTE

This article provides steps for how to delete personal data from the device or service and can be used to support your obligations under the GDPR. If you're looking for general info about GDPR, see the [GDPR section of the Service Trust portal](#).

Control your account data

In-product data stored by Azure Machine Learning experimentation and model management is available for export and deletion through the Azure portal, CLI, SDK, and authenticated REST APIs. Telemetry data can be accessed through the Azure Privacy portal.

In Azure Machine Learning, personal data consists of user information in run history documents and telemetry records of some user interactions with the service.

Delete account data with the REST API

In order to delete data, the following API calls can be made with the HTTP DELETE verb. These are authorized by having an `Authorization: Bearer <arm-token>` header in the request, where `<arm-token>` is the AAD access token for the endpoint `https://management.core.windows.net/` endpoint.

To learn how to get this token and call Azure endpoints, see [Azure REST API documentation](#).

In the examples following, replace the text in {} with the instance names that determine the associated resource.

Delete from a hosting account

```
https://management.azure.com/subscriptions/{subscription-id}/resourceGroups/{resource-group}/providers/Microsoft.MachineLearningModelManagement/accounts/{account-name}?api-version=2017-09-01-preview
```

Delete from the model management service

Model document

Use this call to get a list of models and their IDs:

```
https://{{location}}.modelmanagement.azureml.net/api/subscriptions/{{subscriptionId}}/resourceGroups/{{resourceGroupName}}/accounts/{{accountName}}/models?api-version=2017-09-01-preview"
```

Individual models can be deleted with:

```
https://{{location}}.modelmanagement.azureml.net/api/subscriptions/{{subscriptionId}}/resourceGroups/{{resourceGroupName}}/accounts/{{accountName}}/models/{{modelId}}?api-version=2017-09-01-preview
```

Manifest document

Use this call to get a list of all manifests and their IDs:

```
https://{{location}}.modelmanagement.azureml.net/api/subscriptions/{{subscriptionId}}/resourceGroups/{{resourceGroupName}}/accounts/{{accountName}}/manifests?api-version=2017-09-01-preview
```

Individual manifests can be deleted with:

```
https://{{location}}.modelmanagement.azureml.net/api/subscriptions/{{subscriptionId}}/resourceGroups/{{resourceGroupName}}/accounts/{{accountName}}/manifests/{{manifestId}}?api-version=2017-09-01-preview
```

Service documents

Use this call to get a list of all services and their IDs:

```
https://{{location}}.modelmanagement.azureml.net/api/subscriptions/{{subscriptionId}}/resourceGroups/{{resourceGroupName}}/accounts/{{accountName}}/services?api-version=2017-09-01-preview
```

Individual services can be deleted with:

```
https://{{location}}.modelmanagement.azureml.net/api/subscriptions/{{subscriptionId}}/resourceGroups/{{resourceGroupName}}/accounts/{{accountName}}/services/{{serviceName}}?api-version=2017-09-01-preview
```

Image document

Use this call to get a list of all images and their IDs:

```
https://{{location}}.modelmanagement.azureml.net/api/subscriptions/{{subscriptionId}}/resourceGroups/{{resourceGroupName}}/accounts/{{accountName}}/images?api-version=2017-09-01-preview
```

Individual images can be deleted with:

```
https://{{location}}.modelmanagement.azureml.net/api/subscriptions/{{subscriptionId}}/resourceGroups/{{resourceGroupName}}/accounts/{{accountName}}/images/{{imageId}}?api-version=2017-09-01-preview
```

Delete run history, artifact, and notification data

Run history, artifact, and notification stores for a project are all deleted after deleting the corresponding project

document:

```
https://management.azure.com/subscriptions/{subscription-id}/resourceGroups/{resource-group}/providers/Microsoft.MachineLearningExperimentation/accounts/{account-name}/workspaces/{workspace-name}/projects/{project-name}?api-version=2017-05-01-preview
```

Delete from experimentation account resource provider

Project documents are deleted with:

```
https://management.azure.com/subscriptions/{subscription-id}/resourceGroups/{resource-group}/providers/Microsoft.MachineLearningExperimentation/accounts/{account-name}/workspaces/{workspace-name}/projects/{project-name}?api-version=2017-05-01-preview
```

Workspace documents are deleted with:

```
https://management.azure.com/subscriptions/{subscription-id}/resourceGroups/{resource-group}/providers/Microsoft.MachineLearningExperimentation/accounts/{account-name}/workspaces/{workspace-name}?api-version=2017-05-01-preview
```

The entire experimentation account is deleted with:

```
https://management.azure.com/subscriptions/{subscription-id}/resourceGroups/{resource-group}/providers/Microsoft.MachineLearningExperimentation/accounts/{account-name}?api-version=2017-05-01-preview
```

Export service data with the REST API

In order to export data, the following API calls can be made with the HTTP GET verb. These are authorized by having an `Authorization: Bearer <arm-token>` header in the request, where `<arm-token>` is the AAD access token for the endpoint `https://management.core.windows.net/`

To learn how to get this token and call Azure endpoints, see [Azure REST API documentation](#).

In the examples following, replace the text in {} with the instance names that determine the associated resource.

Export hosting account data

```
https://management.azure.com/subscriptions/{subscriptionId}/resourceGroups/{resourceGroupName}/providers/Microsoft.MachineLearningModelManagement/accounts/{accountName}?api-version=2017-09-01-preview
```

Export model management service data

Model document

Use this call to get a list of models and their IDs:

```
https://{{location}}.modelmanagement.azureml.net/api/subscriptions/{subscriptionId}/resourceGroups/{resourceGroupName}/accounts/{accountName}/models?api-version=2017-09-01-preview"
```

Individual models can be obtained by:

```
https://{{location}}.modelmanagement.azureml.net/api/subscriptions/{{subscriptionId}}/resourceGroups/{{resourceGroupName}}/accounts/{{accountName}}/models/{{modelId}}?api-version=2017-09-01-preview
```

Manifests

Use this call to get a list of all manifests and their IDs:

```
https://{{location}}.modelmanagement.azureml.net/api/subscriptions/{{subscriptionId}}/resourceGroups/{{resourceGroupName}}/accounts/{{accountName}}/manifests?api-version=2017-09-01-preview
```

Individual manifests can be obtained by:

```
https://{{location}}.modelmanagement.azureml.net/api/subscriptions/{{subscriptionId}}/resourceGroups/{{resourceGroupName}}/accounts/{{accountName}}/manifests/{{manifestId}}?api-version=2017-09-01-preview
```

Services

Use this call to get a list of all services and their IDs:

```
https://{{location}}.modelmanagement.azureml.net/api/subscriptions/{{subscriptionId}}/resourceGroups/{{resourceGroupName}}/accounts/{{accountName}}/services?api-version=2017-09-01-preview
```

Individual services can be obtained by:

```
https://{{location}}.modelmanagement.azureml.net/api/subscriptions/{{subscriptionId}}/resourceGroups/{{resourceGroupName}}/accounts/{{accountName}}/services/{{serviceName}}?api-version=2017-09-01-preview
```

Images

Use this call to get a list of all images and their IDs:

```
https://{{location}}.modelmanagement.azureml.net/api/subscriptions/{{subscriptionId}}/resourceGroups/{{resourceGroupName}}/accounts/{{accountName}}/images?api-version=2017-09-01-preview
```

Individual services can be obtained by:

```
https://{{location}}.modelmanagement.azureml.net/api/subscriptions/{{subscriptionId}}/resourceGroups/{{resourceGroupName}}/accounts/{{accountName}}/images/{{imageId}}?api-version=2017-09-01-preview
```

Export compute data

Compute clusters

Use this call to get a list of all compute clusters and their names:

```
https://management.azure.com/subscriptions/{{subscriptionId}}/resourceGroups/{{resourceGroupName}}/providers/Microsoft.MachineLearningServices/workspaces/{{workspaceName}}/computes?api-version=2018-03-01-preview
```

Individual clusters can be obtained by:

```
https://management.azure.com/subscriptions/{{subscriptionId}}/resourceGroups/{{resourceGroupName}}/providers/Microsoft.MachineLearningServices/workspaces/{{workspaceName}}/computes/{{compute-name}}?api-version=2018-03-01-preview
```

Operationalization clusters

Use this call to get a list of all clusters and their names:

```
https://management.azure.com/subscriptions/{subscriptionId}/resourcegroups/{resourceGroupName}/providers/Microsoft.MachineLearningCompute/operationalizationClusters?api-version=2017-06-01-preview
```

Individual clusters can be obtained by:

```
https://management.azure.com/subscriptions/{subscriptionId}/resourcegroups/{resourceGroupName}/providers/Microsoft.MachineLearningCompute/operationalizationClusters/{clusterName}?api-version=2017-06-01-preview
```

Export run history data

Use this call to get a list of all runs and their IDs:

```
https://{{location}}.experiments.azureml.net/history/v1.0/subscriptions/{subscriptionId}/resourceGroups/{resourceGroupName}/providers/Microsoft.MachineLearningExperimentation/accounts/{accountName}/workspaces/{workspaceName}/projects/{ projectName}/runs
```

Use this call to get a list of all experiments and their IDs:

```
https://{{location}}.experiments.azureml.net/history/v1.0/subscriptions/{subscriptionId}/resourceGroups/{resourceGroupName}/providers/Microsoft.MachineLearningExperimentation/accounts/{accountName}/workspaces/{workspaceName}/projects/{ projectName}/experiments
```

Run history items can be obtained by:

```
https://{{location}}.experiments.azureml.net/history/v1.0/subscriptions/{subscriptionId}/resourceGroups/{resourceGroupName}/providers/Microsoft.MachineLearningExperimentation/accounts/{accountName}/workspaces/{workspaceName}/projects/{ projectName}/runs/{runId}
```

Run metrics items can be obtained by:

```
https://{{location}}.experiments.azureml.net/history/v1.0/subscriptions/{subscriptionId}/resourceGroups/{resourceGroupName}/providers/Microsoft.MachineLearningExperimentation/accounts/{accountName}/workspaces/{workspaceName}/projects/{ projectName}/runmetrics
```

Run experiments can be obtained by:

```
https://{{location}}.experiments.azureml.net/history/v1.0/subscriptions/{subscriptionId}/resourceGroups/{resourceGroupName}/providers/Microsoft.MachineLearningExperimentation/accounts/{accountName}/workspaces/{workspaceName}/projects/{ projectName}/experiments/{experimentId}
```

Run history artifacts:

```
https://{{location}}.experiments.azureml.net/history/v1.0/subscriptions/{subscriptionId}/resourceGroups/{resourceGroupName}/providers/Microsoft.MachineLearningExperimentation/accounts/{accountName}/workspaces/{workspaceName}/projects/{ projectName}/runs/{runId}/artifacts
```

Run history artifacts URLs:

```
https://{{location}}.experiments.azureml.net/history/v1.0/subscriptions/{{subscriptionId}}/resourceGroups/{{resourceGroupName}}/providers/Microsoft.MachineLearningExperimentation/accounts/{{accountName}}/workspaces/{{workspaceName}}/projects/{{ projectName}}/runs/{{runId}}/artifacturi?name={{artifactName}}
```

Export artifacts

Use this call to get a list of assets and their names:

```
https://{{location}}.experiments.azureml.net/artifact/v1.0/subscriptions/{{subscriptionId}}/resourceGroups/{{resourceGroupName}}/providers/Microsoft.MachineLearningExperimentation/accounts/{{accountName}}/workspaces/{{workspaceName}}/projects/{{ projectName}}/assets
```

Use this call to get a list of artifacts and their paths:

```
https://{{location}}.experiments.azureml.net/artifact/v1.0/subscriptions/{{subscriptionId}}/resourceGroups/{{resourceGroupName}}/providers/Microsoft.MachineLearningExperimentation/accounts/{{accountName}}/workspaces/{{workspaceName}}/projects/{{ projectName}}/artifacts/origins/{{origin}}/containers/{{runId}}
```

Artifact contents

```
https://{{location}}.experiments.azureml.net/artifact/v1.0/subscriptions/{{subscriptionId}}/resourceGroups/{{resourceGroupName}}/providers/Microsoft.MachineLearningExperimentation/accounts/{{accountName}}/workspaces/{{workspaceName}}/projects/{{ projectName}}/artifacts/contentinfo/ExperimentRun/{{runId}}/{{artifactPath}}
```

Artifact documents

```
https://{{location}}.experiments.azureml.net/history/v1.0/subscriptions/{{subscriptionId}}/resourceGroups/{{resourceGroupName}}/providers/Microsoft.MachineLearningExperimentation/accounts/{{accountName}}/workspaces/{{workspaceName}}/projects/{{ projectName}}/runs/{{runId}}/artifacts
```

Assets

```
https://{{location}}.experiments.azureml.net/artifact/v1.0/subscriptions/{{subscriptionId}}/resourceGroups/{{resourceGroupName}}/providers/Microsoft.MachineLearningExperimentation/accounts/{{accountName}}/workspaces/{{workspaceName}}/projects/{{ projectName}}/assets/name/{{name}}
```

Export notifications

1. Go to the [Users section of the Azure portal](#), and then select a user from the **Name** column.
2. Note the **Object ID**, and use it in the following call:

```
https://{{location}}.experiments.azureml.net/notification/v1.0/subscriptions/{{subscriptionId}}/resourceGroups/{{resourceGroupName}}/providers/Microsoft.MachineLearningExperimentation/accounts/{{accountName}}/workspaces/{{workspaceName}}/projects/{{ projectName}}/users/{{objectId}}/jobs
```

Export experimentation account information

Experimentation account information

```
https://management.azure.com/subscriptions/{{subscriptionId}}/resourceGroups/{{resourceGroupName}}/providers/Microsoft.MachineLearningExperimentation/accounts/{{accountName}}?api-version=2017-05-01-preview
```

Workspace information

```
https://management.azure.com/subscriptions/{subscriptionId}/resourceGroups/{resourceGroupName}/providers/Microsoft.MachineLearningExperimentation/accounts/{accountName}/workspaces/{workspaceName}?api-version=2017-05-01-preview
```

Project information

```
https://management.azure.com/subscriptions/{subscriptionId}/resourceGroups/{resourceGroupName}/providers/Microsoft.MachineLearningExperimentation/accounts/{accountName}/workspaces/{workspaceName}/projects/{ projectName}?api-version=2017-05-01-preview
```

Quickstart: Install and get started with Azure Machine Learning service

12/11/2018 • 9 minutes to read • [Edit Online](#)

NOTE

This article is deprecated. Support for earlier versions of this service will end incrementally. [View the support timeline](#). Start using the latest version with this [quickstart](#).

Azure Machine Learning service (preview) is an integrated, end-to-end data science and advanced analytics solution. It helps professional data scientists prepare data, develop experiments, and deploy models at cloud scale.

This quickstart shows you how to:

- Create service accounts for Azure Machine Learning service
- Install and log in to Azure Machine Learning Workbench.
- Create a project in Workbench
- Run a script in that project
- Access the command-line interface (CLI)

As part of the Microsoft Azure portfolio, Azure Machine Learning service requires an Azure subscription. If you don't have an Azure subscription, create a [free account](#) before you begin.

Additionally, you must have adequate permissions to create assets such as Resource Groups, Virtual Machines, and so on.

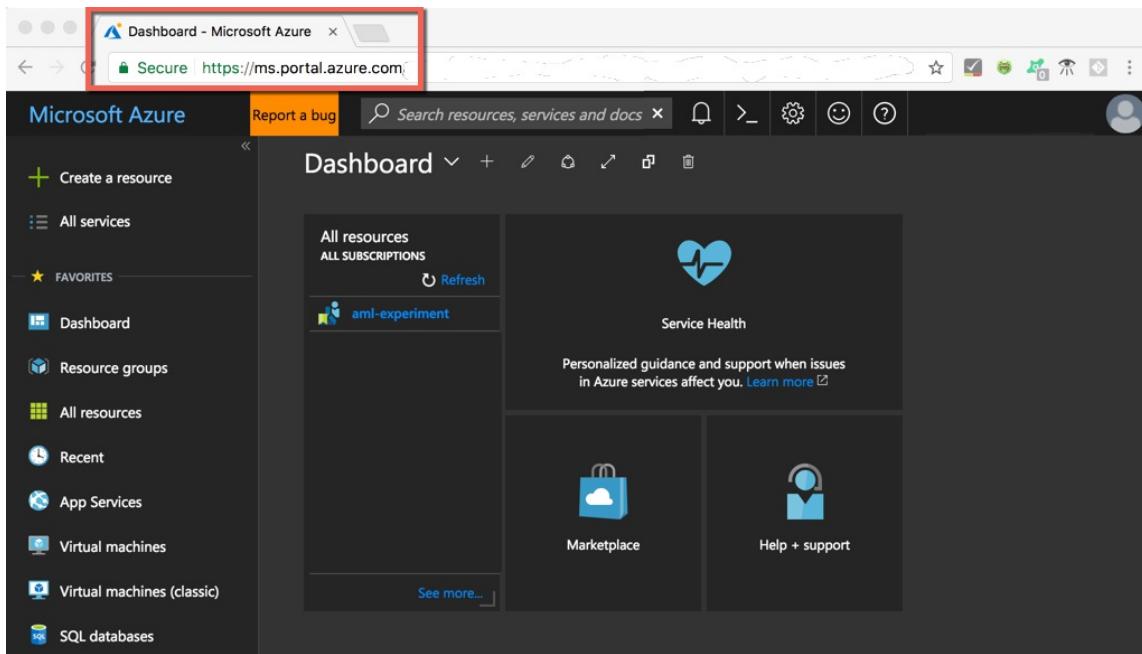
You can install the Azure Machine Learning Workbench application on the following operating systems:

- Windows 10 or Windows Server 2016
- macOS Sierra or High Sierra

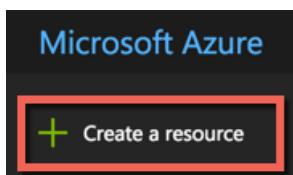
Create Azure Machine Learning service accounts

Use the Azure portal to provision your Azure Machine Learning accounts:

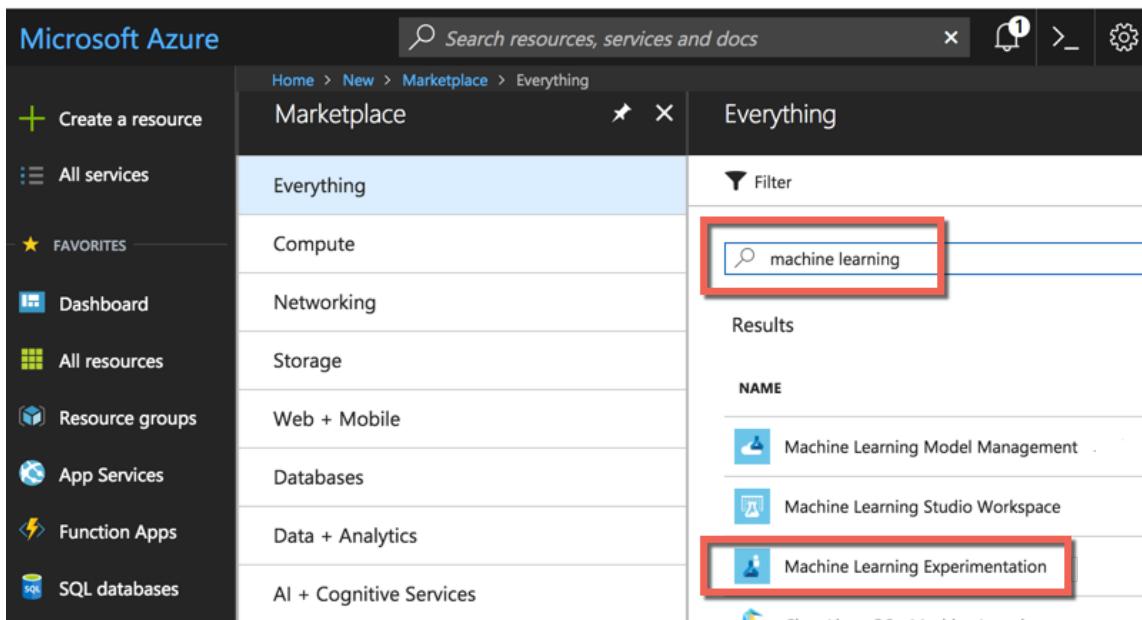
1. Sign in to the [Azure portal](#) using the credentials for the Azure subscription you'll use. If you don't have an Azure subscription, create a [free account](#) now.



2. Select the **Create a resource** button (+) in the upper-left corner of the portal.



3. Enter **Machine Learning** in the search bar. Select the search result named **Machine Learning Experimentation**.



4. In the **Machine Learning Experimentation** pane, scroll to the bottom and select **Create** to begin defining your experimentation account.



5. In the **ML Experimentation** pane, configure your Machine Learning Experimentation account.

Setting	Suggested Value for Tutorial	Description
Experimentation account name	<i>Unique name</i>	Enter a unique name that identifies your account. You can use your own name, or a departmental or project name that best identifies the experiment. The name should be 2 to 32 characters. It should include only alphanumeric characters and the dash (-) character.
Subscription	<i>Your subscription</i>	Choose the Azure subscription that you want to use for your experiment. If you have multiple subscriptions, choose the appropriate subscription in which the resource is billed.
Resource group	<i>Your resource group</i>	Use an existing resource group in your subscription, or enter a name to create a new resource group for this experimentation account.
Location	<i>The region closest to your users</i>	Choose the location closest to your users and the data resources.
Number of seats	2	Enter the number of seats. Learn how seating impacts pricing . For this Quickstart, you only need two seats. Seats can be added or removed as needed in the Azure portal.
Storage account	<i>Unique name</i>	Select Create new and provide a name to create an Azure storage account . The name should be 3 to 24 characters, and should include only alphanumeric characters. Alternatively, select Use existing and select your existing storage account from the drop-down list. The storage account is required and is used to hold project artifacts and run history data.
Workspace for Experimentation account	IrisGarden (name used in tutorials)	Provide a name for a workspace for this account. The name should be 2 to 32 characters. It should include only alphanumeric characters and the dash (-) character. This workspace contains the tools you need to create, manage, and publish experiments.
Assign owner for the workspace	<i>Your account</i>	Select your own account as the workspace owner.

SETTING	SUGGESTED VALUE FOR TUTORIAL	DESCRIPTION
Create Model Management account	check	<p>Create a Model Management account now so that this resource is available when you want to deploy and manage your models as real-time web services.</p> <p>While optional, we recommend creating the Model Management account at the same time as the Experimentation account.</p>
Account name	<i>Unique name</i>	Choose a unique name that identifies your Model Management account. You can use your own name, or a departmental or project name that best identifies the experiment. The name should be 2 to 32 characters. It should include only alphanumeric characters and the dash (-) character.
Model Management pricing tier	DEVTEST	Select No pricing tier selected to specify the pricing tier for your new Model Management account. For cost savings, select the DEVTEST pricing tier if it's available on your subscription (limited availability). Otherwise, select the S1 pricing tier. Click Select to save the pricing tier selection.
Pin to dashboard	<i>check</i>	Select the Pin to dashboard option to allow easy tracking of your Machine Learning Experimentation account on the front dashboard page of the Azure portal.

ML Experimentation

Machine Learning Experimentation

* Experimentation account name

* Subscription

* Resource group
 Create new Use existing

* Location

* Number of seats. First two are free. ⓘ

* Storage account ⓘ
 Create new Use existing

* Workspace for Experimentation account ⓘ

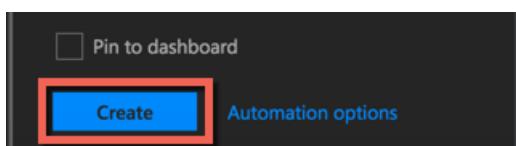
* Assign owner for the workspace >

Create Model Management account ⓘ

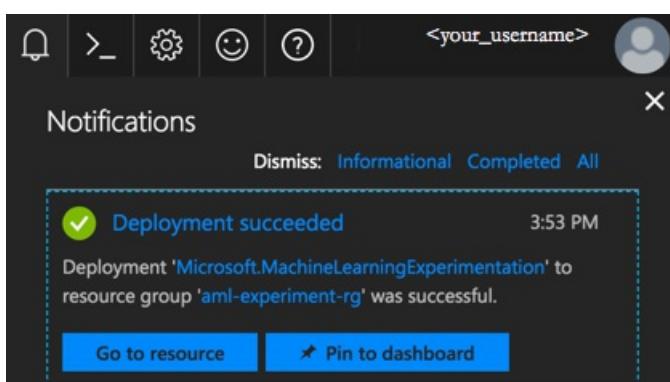
* Account name

* Model Management pricing tier ⓘ >
No pricing tier selected

6. Select **Create** to begin the creation process of the Experimentation account along with the Model Management account.



It can take a few moments to create an account. You can check on the status of the deployment process by clicking the Notifications icon (bell) on the Azure portal toolbar.



Install and log in to Workbench

Azure Machine Learning Workbench is available for Windows or macOS. See the list of [supported platforms](#).

WARNING

The installation might take around 30 minutes to complete.

1. Download and launch the latest Workbench installer.

IMPORTANT

Download the installer fully on disk, and then run it from there. Do not run it directly from your browser's download widget.

On Windows:

- A. Download [AmlWorkbenchSetup.msi](#).
- B. Double-click on the downloaded installer in File Explorer.

On macOS:

- A. Download [AmlWorkbench.dmg](#).
- B. Double-click on the downloaded installer in Finder.

2. Follow the on-screen instructions in your installer to completion.

The installation might take around 30 minutes to complete.

INSTALLATION PATH TO AZURE MACHINE LEARNING WORKBENCH	
Windows	C:\Users\<user>\AppData\Local\AmlWorkbench
macOS	/Applications/Azure ML Workbench.app

The installer will download and set up all the necessary dependencies, such as Python, Miniconda, and other related libraries. This installation also includes the Azure cross-platform command-line tool, or Azure CLI.

3. Launch Workbench by selecting the **Launch Workbench** button on the last screen of the installer.

If you closed the installer:

- On Windows, launch it using the **Machine Learning Workbench** desktop shortcut.
- On macOS, select **Azure ML Workbench** in Launchpad.

4. On the first screen, select **Sign in with Microsoft** to authenticate with the Azure Machine Learning Workbench. Use the same credentials you used in the Azure portal to create the Experimentation and Model Management accounts.

Once you are signed in, Workbench uses the first Experimentation account it finds in your Azure subscriptions, and displays all workspaces and projects associated with that account.

TIP

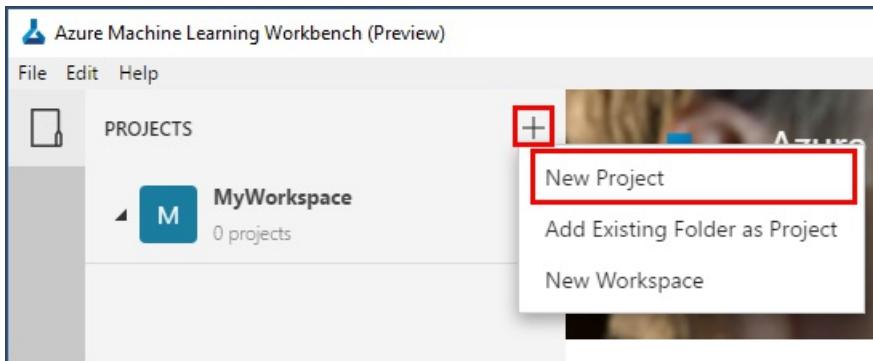
You can switch to a different Experimentation account using the icon in the lower-left corner of the Workbench application window.

Create a project in Workbench

In Azure Machine Learning, a project is the logical container for all the work being done to solve a problem. It maps to a single folder on your local disk, and you can add any files or subfolders to it.

Here, we are creating a new Workbench project using a template that includes the [Iris flower dataset](#). The tutorials that follow this quickstart depend on this data to build a model that predicts the type of iris based on some of its physical characteristics.

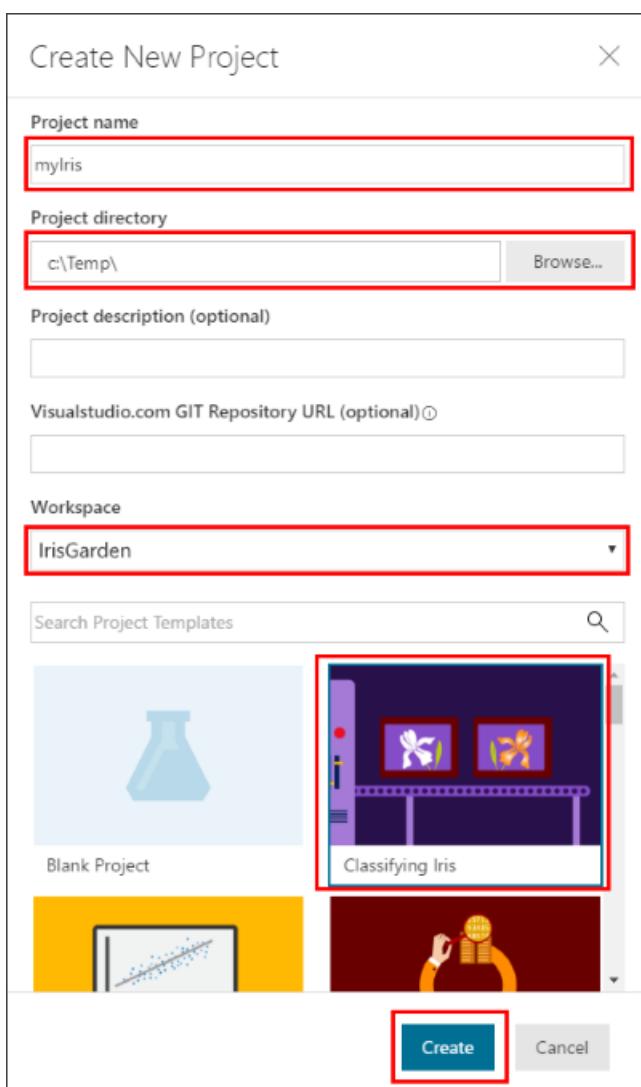
1. With Azure Machine Learning Workbench open, select the plus sign (+) in the **PROJECTS** pane and choose **New Project**.



2. Fill out of the form fields and select the **Create** button to create a new project in the Workbench.

FIELD	SUGGESTED VALUE FOR TUTORIAL	DESCRIPTION
Project name	myIris	Enter a unique name that identifies your account. You can use your own name, or a departmental or project name that best identifies the experiment. The name should be 2 to 32 characters. It should include only alphanumeric characters and the dash (-) character.
Project directory	c:\Temp\	Specify the directory in which the project is created.
Project description	<i>leave blank</i>	Optional field useful for describing the projects.
Visualstudio.com GIT Repository URL	<i>leave blank</i>	Optional field. A project can optionally be associated with a Git repository on Azure DevOps for source control and collaboration. Learn how to set that up..

FIELD	SUGGESTED VALUE FOR TUTORIAL	DESCRIPTION
Selected workspace	IrisGarden (if it exists)	Choose a workspace that you have created for your Experimentation account in the Azure portal. If you followed the Quickstart, you should have a workspace by the name IrisGarden. If not, select the one you created when you created your Experimentation account or any other you want to use.
Project template	Classifying Iris	Templates contain scripts and data you can use to explore the product. This template contains the scripts and data you need for this quickstart and other tutorials in this documentation site.



A new project is created and the project dashboard opens with that project. At this point, you can explore the project home page, data sources, notebooks, and source code files.

TIP

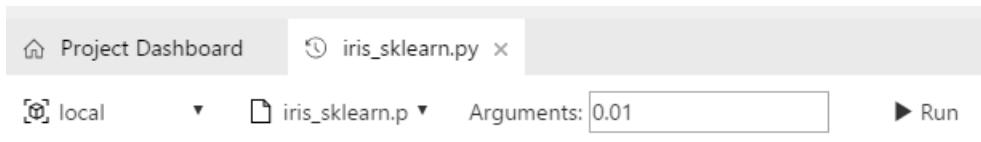
You can configure Workbench to work with a Python IDE for a smooth data science development experience. Then, you can interact with your project in the IDE. [Learn how](#).

Run a Python script

Now, you can run the **iris_sklearn.py** script on your local computer. This script is included by default with the **Classifying Iris** project template. The script builds a [logistic regression](#) model using the popular Python [scikit-learn](#) library.

1. In the command bar at the top of the **Project Dashboard** page, select **local** as the execution target and select **iris_sklearn.py** as the script to run. These values are preselected by default.

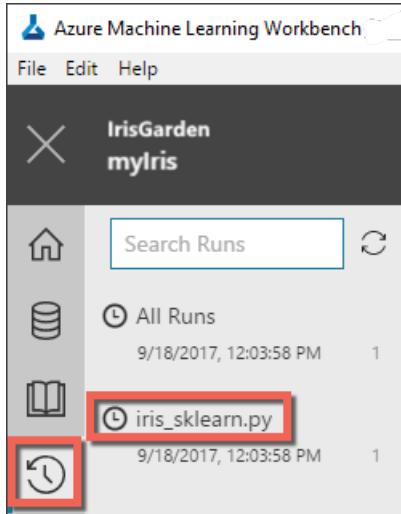
There are other files included in the sample that you can check out later, but for this quickstart we are only interested in **iris_sklearn.py**.



2. In the **Arguments** text box, enter **0.01**. This number corresponds to the regularization rate, and is used in the script to configure the logistic regression model.
3. Select **Run** to start the execution of the script on your computer. The **iris_sklearn.py** job immediately appears in the **Jobs** panel on the right so you can monitor the script's execution.

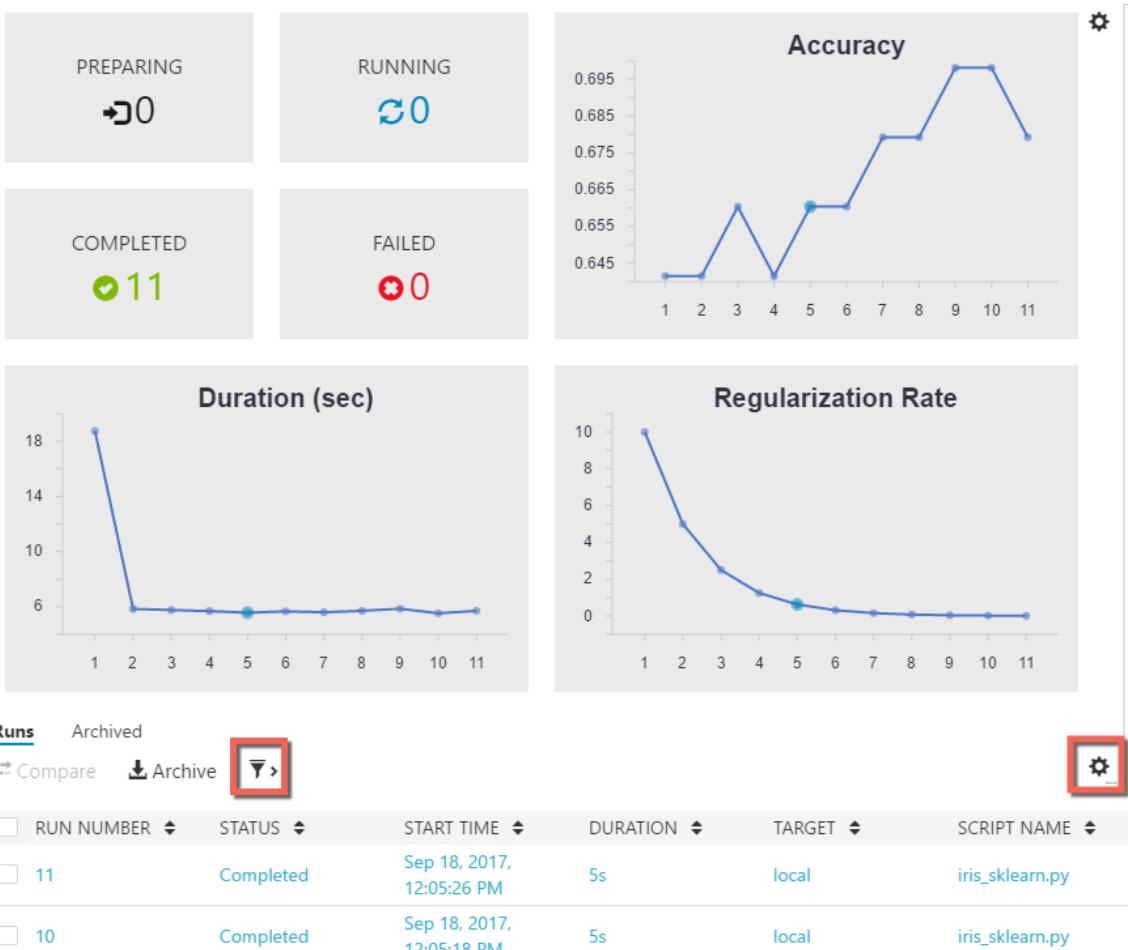
Congratulations! You've successfully run a Python script in Azure Machine Learning Workbench.

4. Repeat steps 2 - 3 several times using different argument values ranging from **0.001** to **10** (for example, using powers of 10). Each run appears in the **Jobs** pane.
5. Inspect the run history by selecting the **Runs** view and then **iris_sklearn.py** in the Runs list.



This view shows every run that was executed on **iris_sklearn.py**. The run history dashboard also displays the top metrics, a set of default graphs, and a list of metrics for each run.

6. You can customize this view by sorting, filtering, and adjusting the configurations using the gear and filter icons.



7. Select a completed run in the Jobs pane to see a detailed view for that specific execution. Details include additional metrics, the files that it produced, and other potentially useful logs.

Start the CLI

The Azure Machine Learning command-line interface (CLI) is also installed. The CLI interface allows you to access and interact with your Azure Machine Learning service using the `az` commands to perform all tasks required for an end-to-end data science workflow. [Learn more](#).

You can launch the Azure Machine Learning CLI from the Workbench's toolbar using **File → Open Command Prompt**.

You can get help on commands in the Azure Machine Learning CLI using the `--help` argument.

```
az ml --help
```

Clean up resources

IMPORTANT

The resources you created can be used as prerequisites to other Azure Machine Learning service tutorials and how-to articles.

If you don't plan to use the resources you created here, delete them so you don't incur any charges.

1. In the Azure portal, select **Resource groups** on the far left.

The screenshot shows the Microsoft Azure portal interface. The left sidebar has a 'Resource groups' item selected, which is highlighted with a red box. The main content area shows the 'newacct' resource group details. At the top right of the content area, there is a 'Delete resource group' button, also highlighted with a red box. The 'Essentials' section displays basic information like Subscription name, Deployment status, and Resource ID. Below that is a table listing resources, with one entry for 'newacct' (Azure Cosmos DB account) located in South Central US.

NAME	TYPE	LOCATION
newacct	Azure Cosmos DB account	South Central US

2. From the list, select the resource group you created.
3. Select **Delete resource group**.
4. Enter the resource group name, and then select **Delete**.

Next steps

You have now created the necessary Azure Machine Learning accounts and installed the Azure Machine Learning Workbench application. You have also created a project, ran a script, and explored the run history of the script.

For a more in-depth experience of this workflow, including how to deploy your Iris model as a web service, follow the full-length *Classifying Iris* tutorial. The tutorial contains detailed steps for [data preparation](#), [experimentation](#), and [model management](#).

[Tutorial: Classifying Iris \(Part 1\)](#)

NOTE

While you have created your model management account, your environment is not set up for deploying web services yet. Learn how to set up your [deployment environment](#).

Tutorial 1: Classify Iris - Preparing the data

12/11/2018 • 7 minutes to read • [Edit Online](#)

NOTE

This article is deprecated. Support for earlier versions of this service will end incrementally. [View the support timeline](#). Start using the latest version with this [quickstart](#).

Azure Machine Learning service (preview) is an integrated, end-to-end data science and advanced analytics solution for professional data scientists to prepare data, develop experiments, and deploy models at cloud scale.

This tutorial is **part one of a three-part series**. In this tutorial, you walk through the basics of Azure Machine Learning service (preview) and learn how to:

- Create a project in Azure Machine Learning Workbench
- Create a data preparation package
- Generate Python/PySpark code to invoke a data preparation package

This tutorial uses the timeless [Iris flower data set](#).

IMPORTANT

Azure Machine Learning service is currently in preview. Previews are made available to you on the condition that you agree to the [supplemental terms of use](#). Some aspects of this feature may change prior to general availability (GA).

Prerequisites

If you don't have an Azure subscription, create a [free account](#) before you begin.

To complete this tutorial, you must have:

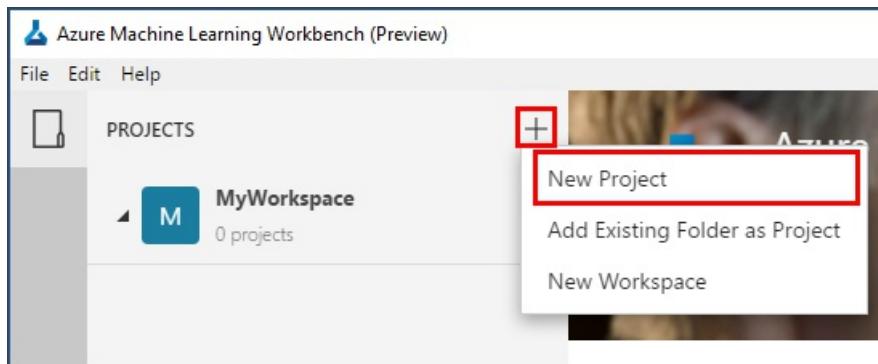
- An Azure Machine Learning Experimentation account
- Azure Machine Learning Workbench installed

If you don't have these prerequisites already, follow the steps in the [Quickstart: Install and start](#) article to set up your accounts and install the Azure Machine Learning Workbench application.

Create a new project in Workbench

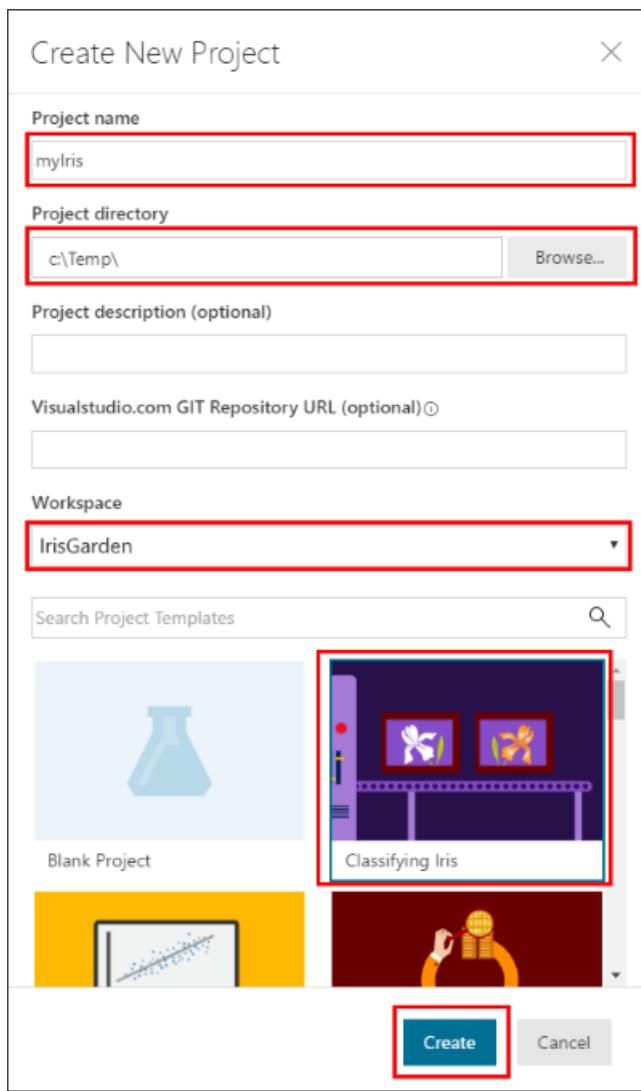
If you followed the steps in the [Quickstart: Install and start](#) article you should already have this project and can skip to the next section.

1. Open the Azure Machine Learning Workbench app, and log in if needed.
 - On Windows, launch it using the **Machine Learning Workbench** desktop shortcut.
 - On macOS, select **Azure ML Workbench** in Launchpad.
2. Select the plus sign (+) in the **PROJECTS** pane and choose **New Project**.



3. Fill out of the form fields and select the **Create** button to create a new project in the Workbench.

FIELD	SUGGESTED VALUE FOR TUTORIAL	DESCRIPTION
Project name	myIris	Enter a unique name that identifies your account. You can use your own name, or a departmental or project name that best identifies the experiment. The name should be 2 to 32 characters. It should include only alphanumeric characters and the dash (-) character.
Project directory	c:\Temp\	Specify the directory in which the project is created.
Project description	<i>leave blank</i>	Optional field useful for describing the projects.
Visualstudio.com GIT Repository URL	<i>leave blank</i>	Optional field. You can associate a project with a Git repository on Azure DevOps for source control and collaboration. Learn how to set that up.
Selected workspace	IrisGarden (if it exists)	Choose a workspace that you have created for your Experimentation account in the Azure portal. If you followed the Quickstart, you should have a workspace by the name IrisGarden. If not, select the one you created when you created your Experimentation account or any other you want to use.
Project template	Classifying Iris	Templates contain scripts and data you can use to explore the product. This template contains the scripts and data you need for this quickstart and other tutorials in this documentation site.



A new project is created and the project dashboard opens with that project. At this point, you can explore the project home page, data sources, notebooks, and source code files.

Project Dashboard

myIris

LOCATION C:/temp/

Created on 4/2/2018, 10:15:05 AM

Run

Classifying Iris

This is a companion sample project of the Azure Machine Learning QuickStart and Tutorials. Using the timeless Iris flower dataset, it walks you through the basics of preparing dataset, creating a model and deploying it as a web service.

QuickStart

Select local as the execution environment, and `iris_sklearn.py` as the script, and click Run button. You can also set the Regularization Rate by entering `0.01` in the Arguments control. Changing the Regularization Rate has an impact on the accuracy of the model, giving interesting results to explore.

Exploring results

After running, you can check out the results in Run History. Exploring the Run History will allow you to see the correlation between the

Create a data preparation package

Next, you can explore and start preparing the data in Azure Machine Learning Workbench. Each transformation you perform in Workbench is stored in a JSON format in a local data preparation package (*.dprep file). This data preparation package is the primary container for your data preparation work in Workbench.

This data preparation package can be handed off later to a runtime, such as local-C#/CoreCLR, Scala/Spark, or Scala/HDI.

1. Select the folder icon to open the Files view, then select **iris.csv** to open that file.

This file contains a table with 5 columns and 50 rows. Four columns are numerical feature columns. The fifth column is a string target column. None of the columns have header names.

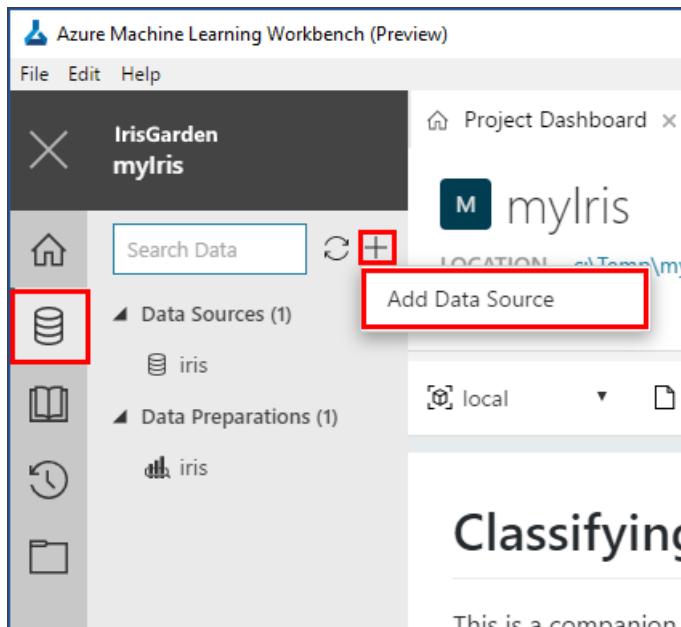
The screenshot shows the 'Files' view in the Azure Machine Learning Workbench. The left sidebar has icons for Home, Project Dashboard, and a red-highlighted Folder icon. The main area shows a list of files under the project 'myIris'. The file 'iris.csv' is highlighted with a red box. The content of 'iris.csv' is listed below:

Content
5.1,3.5,1.4,0.2,Iris-setosa
4.9,3.0,1.4,0.2,Iris-setosa
4.7,3.2,1.3,0.2,Iris-setosa
4.6,3.1,1.5,0.2,Iris-setosa
5.0,3.6,1.4,0.2,Iris-setosa
5.4,3.9,1.7,0.4,Iris-setosa
4.6,3.4,1.4,0.3,Iris-setosa
5.0,3.4,1.5,0.2,Iris-setosa
4.4,2.9,1.4,0.2,Iris-setosa
4.9,3.1,1.5,0.1,Iris-setosa
5.4,3.7,1.5,0.2,Iris-setosa
4.8,3.4,1.6,0.2,Iris-setosa
4.8,3.0,1.4,0.1,Iris-setosa
4.3,3.0,1.1,0.1,Iris-setosa
5.8,4.0,1.2,0.2,Iris-setosa
5.7,4.4,1.5,0.4,Iris-setosa
5.4,3.9,1.3,0.4,Iris-setosa
5.1,3.5,1.4,0.3,Iris-setosa
5.7,3.8,1.7,0.3,Iris-setosa
5.1,3.8,1.5,0.3,Iris-setosa
5.4,3.4,1.7,0.2,Iris-setosa
5.1,3.7,1.5,0.4,Iris-setosa
4.6,3.6,1.0,0.2,Iris-setosa

NOTE

Do not include data files in your project folder, particularly when the file size is large. Because the **iris.csv** data file is tiny, it was included in this template for demonstration purposes. For more information, see [How to read and write large data files](#).

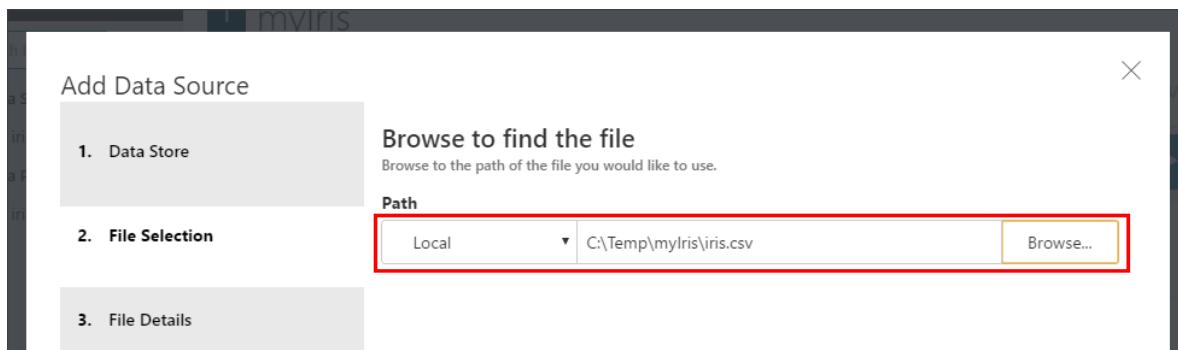
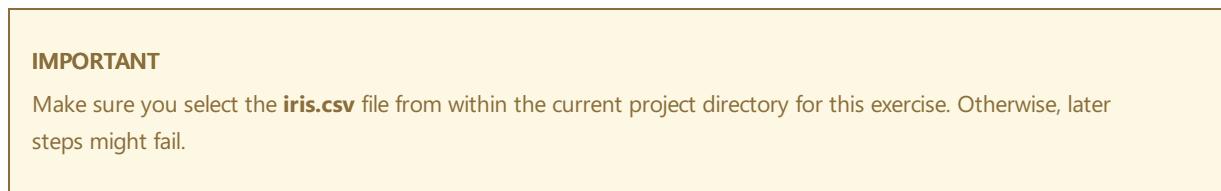
2. In the **Data view**, select the plus sign (+) to add a new data source. The **Add Data Source** page opens.



3. Select **Text Files(*.csv, *.json, *.txt, ...)** and click **Next**.



4. Browse to the file **iris.csv**, and click **Finish**. This will use default values for parameters such as the separator and data types.



5. A new file named **iris-1.dssource** is created. The file is named uniquely with "-1" because the sample project already comes with an unnumbered **iris.dssource** file.

The file opens, and the data is shown. A series of column headers, from **Column1** to **Column5**, is automatically added to this data set. Scroll to the bottom and notice that the last row of the data set is empty. The row is empty because there is an extra line break in the CSV file.

Project Dashboard iris.csv iris-1 X

Metrics Prepare

Columns: 5
Rows: 151

#	Column1	#	Column2	#	Column3	#	Column4	abc	Column5
1	5.1		3.5		1.4		0.2		Iris-setosa
2	4.9		3		1.4		0.2		Iris-setosa
3	4.7		3.2		1.3		0.2		Iris-setosa
4	4.6		3.1		1.5		0.2		Iris-setosa
5	5		3.6		1.4		0.2		Iris-setosa
6	5.4		3.9		1.7		0.4		Iris-setosa
7	4.6		3.4		1.4		0.3		Iris-setosa
8	5		3.4		1.5		0.2		Iris-setosa
9	4.4		2.9		1.4		0.2		Iris-setosa
10	4.9		3.1		1.5		0.1		Iris-setosa

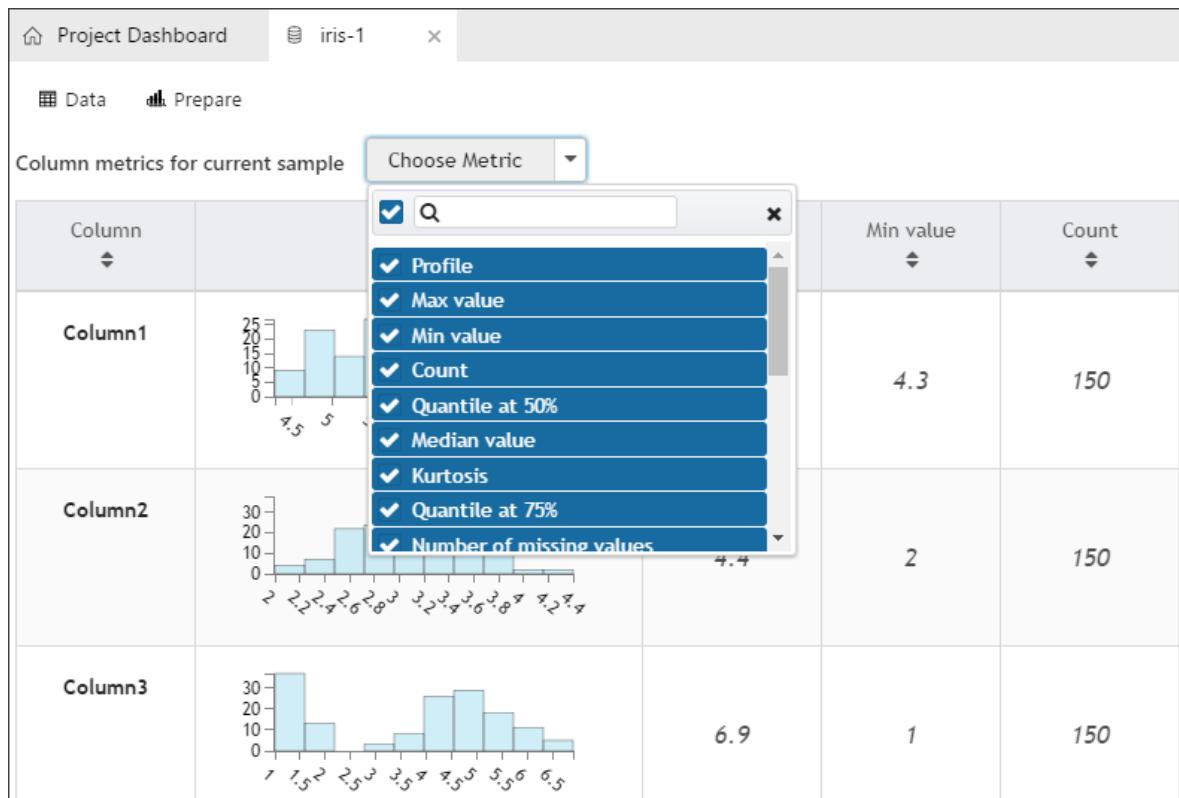
6. Select the **Metrics** button. Histograms are generated and displayed.

You can switch back to the data view by selecting the **Data** button.

Project Dashboard iris.csv iris-1 X

Metrics Prepare

7. Observe the histograms. A complete set of statistics has been calculated for each column.



8. Begin creating a data preparation package by selecting the **Prepare** button. The **Prepare** dialog box opens.

The sample project contains a **iris.dprep** data preparation file that is selected by default.

Prepare

X

Data Preparation Package

iris.dprep

9. Create a new data preparation package by selecting + **New Data Preparation Package** from the drop-down menu.

Prepare

X

Data Preparation Package

+ New Data Preparation Package
✓ iris.dprep

10. Enter a new value for the package name (use **iris-1**) and then select **OK**.

A new data preparation package named **iris-1.dprep** is created and opened in the data preparation editor.

Prepare

X

Data Preparation Package

+ New Data Preparation Package

Data Preparation Package Name

iris-1

Now, let's do some basic data preparation.

11. Select each column header to make the header text editable. Then, rename each column as follows:

In order, enter **Sepal Length**, **Sepal Width**, **Petal Length**, **Petal Width**, and **Species** for the five columns respectively.

#	Sepal Leng..	Sep	Column3	Column4
1	5.1	3.5	1.4	0
2	4.9	3	1.4	0
3	4.7	3.2	1.3	0
4	4.6	3.1	1.5	0

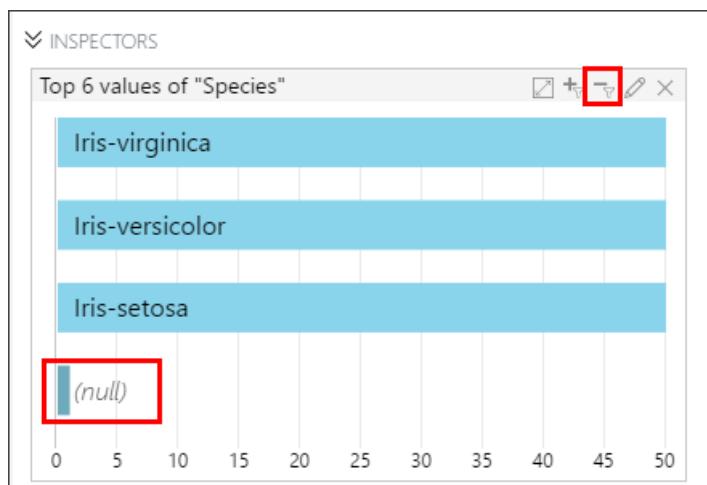
12. Count distinct values:

- Select the **Species** column
- Right-click to select it.
- Select **Value Counts** from the drop-down menu.

The **Inspectors** pane opens below the data. A histogram with four bars appears. The target column has four distinct values: **Iris-virginica**, **Iris-versicolor**, **Iris-setosa**, and a **(null)** value.

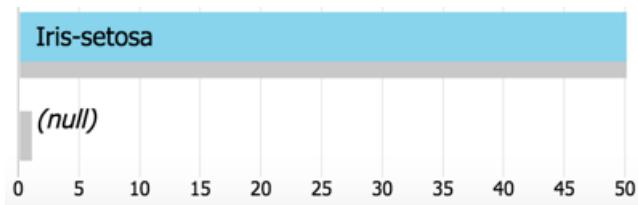
The screenshot shows the Dataflows pane with the iris dataset. The Species column is selected, and a context menu is open, listing various data manipulation options. The 'Value Counts' option is highlighted with a red box.

	abc Sepal Length	abc Sepal Width	abc Petal Length	abc Petal Width	abc Species	
1	5.1	3.5	1.4	0.2	Iris-setosa	Derive Column by Example
2	4.9	3.0	1.4	0.2	Iris-setosa	Split Column by Example
3	4.7	3.2	1.3	0.2	Iris-setosa	Expand JSON
4	4.6	3.1	1.5	0.2	Iris-setosa	Duplicate Column
5	5.0	3.6	1.4	0.2	Iris-setosa	Text Clustering
6	5.4	3.9	1.7	0.4	Iris-setosa	Replace Values
7	4.6	3.4	1.4	0.3	Iris-setosa	Trim String
8	5.0	3.4	1.5	0.2	Iris-setosa	Replace NA Values
9	4.4	2.9	1.4	0.2	Iris-setosa	Replace Missing Values
10	4.9	3.1	1.5	0.1	Iris-setosa	Replace Errors
11	5.4	3.7	1.5	0.2	Iris-setosa	Rename Column
12	4.8	3.4	1.6	0.2	Iris-setosa	Remove Column
13	4.8	3.0	1.4	0.1	Iris-setosa	Keep Column
14	4.3	3.0	1.1	0.1	Iris-setosa	Convert Field Type to Numeric
15	5.8	4.0	1.2	0.2	Iris-setosa	Convert Field Type to Date
16	5.7	4.4	1.5	0.4	Iris-setosa	Convert Field Type to Boolean
17	5.4	3.9	1.3	0.4	Iris-setosa	Filter Column
18	5.1	3.5	1.4	0.3	Iris-setosa	Remove Duplicates
19	5.7	3.8	1.7	0.3	Iris-setosa	Sort
20	5.1	3.8	1.5	0.3	Iris-setosa	Add Column (Script)
21	5.4	3.4	1.7	0.2	Iris-setosa	Value Counts
22	5.1	3.7	1.5	0.4	Iris-setosa	
23	4.6	3.6	1.0	0.2	Iris-setosa	
24	5.1	2.2	1.7	0.5	Iris-setosa	

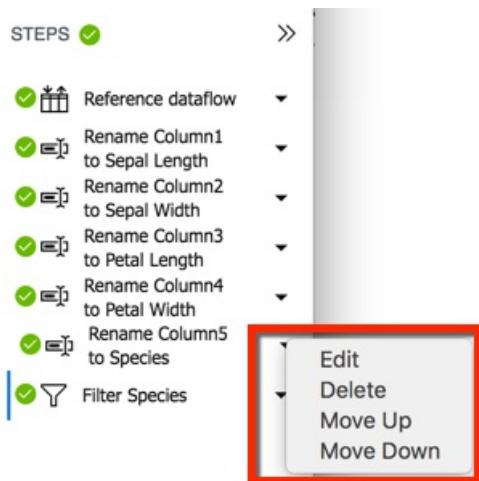


- To filter out the null values, select the "(null)" bar and then select the minus sign (-).

Then, the (null) row turns gray to indicate that it was filtered out.



14. Take notice of the individual data preparation steps that are detailed in the **STEPS** pane. As you renamed the columns and filtered the null value rows, each action was recorded as a data preparation step. You can edit individual steps to adjust their settings, reorder the steps, and remove steps.



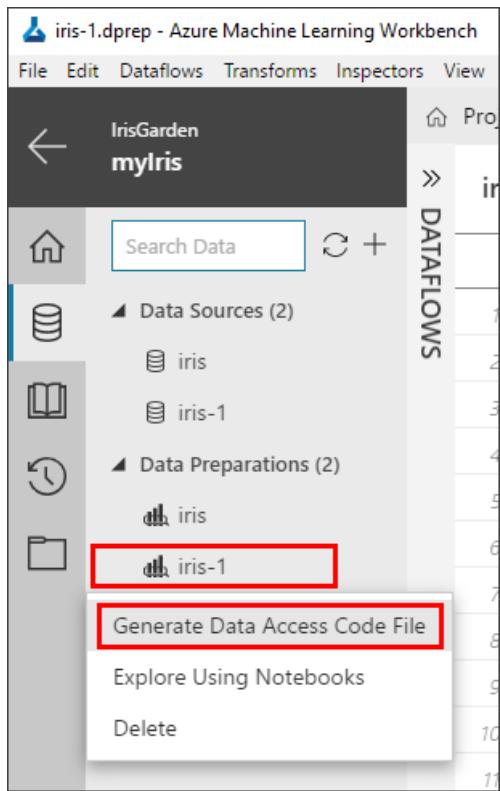
15. Close the data preparation editor. Select the **x** icon on the **iris-1** tab with the graph icon to close the tab. Your work is automatically saved into the **iris-1.dprep** file shown under the **Data Preparations** heading.



Generate Python/PySpark code to invoke a data preparation package

The output of a data preparation package can be explored directly in Python or in a Jupyter Notebook. The packages can be executed across multiple runtimes including local Python, Spark (including in Docker), and HDInsight.

1. Find the **iris-1.dprep** file under the Data Preparations tab.
2. Right-click the **iris-1.dprep** file, and select **Generate Data Access Code File** from the context menu.



A new file named **iris-1.py** opens with the following lines of code to invoke the logic you created as a data preparation package:

```
# Use the Azure Machine Learning data preparation package
from azureml.dataprep import package

# Use the Azure Machine Learning data collector to log various metrics
from azureml.logging import get_azureml_logger
logger = get_azureml_logger()

# This call will load the referenced package and return a DataFrame.
# If run in a PySpark environment, this call returns a
# Spark DataFrame. If not, it will return a Pandas DataFrame.
df = package.run('iris-1.dprep', dataflow_idx=0)

# Remove this line and add code that uses the DataFrame
df.head(10)
```

Depending on the context in which this code is run, `df` represents a different kind of DataFrame:

- When executing on a Python runtime, a [pandas DataFrame](#) is used.
- When executing in a Spark context, a [Spark DataFrame](#) is used.

To learn more about how to prepare data in Azure Machine Learning Workbench, see the [Get started with data preparation](#) guide.

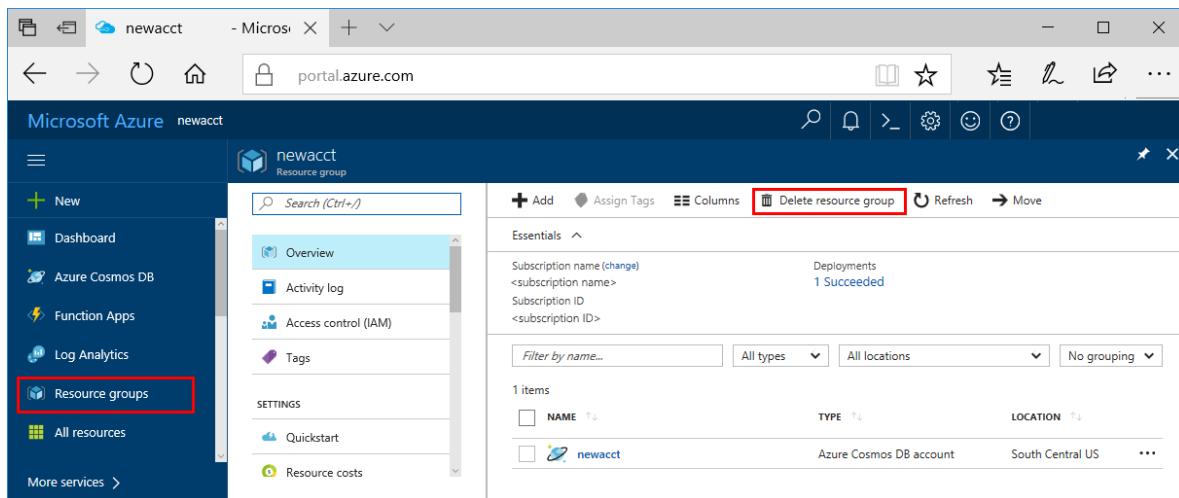
Clean up resources

IMPORTANT

The resources you created can be used as prerequisites to other Azure Machine Learning service tutorials and how-to articles.

If you don't plan to use the resources you created here, delete them so you don't incur any charges.

1. In the Azure portal, select **Resource groups** on the far left.



The screenshot shows the Microsoft Azure portal interface. On the left, there's a sidebar with various service links: New, Dashboard, Azure Cosmos DB, Function Apps, Log Analytics, Resource groups (which is selected and highlighted with a red box), All resources, and More services. The main content area is titled 'newacct Resource group'. It has tabs for Overview, Activity log, Access control (IAM), and Tags. Below these are sections for SETTINGS (Quickstart, Resource costs) and Essentials (Subscription name, Deployment status). A table lists one item: 'newacct' (Azure Cosmos DB account, South Central US). At the top right, there are buttons for Add, Assign Tags, Columns, Delete resource group (highlighted with a red box), Refresh, and Move. The URL in the browser bar is 'portal.azure.com'.

2. From the list, select the resource group you created.
3. Select **Delete resource group**.
4. Enter the resource group name, and then select **Delete**.

Next steps

In this tutorial, you used Azure Machine Learning Workbench to:

- Create a new project
- Create a data preparation package
- Generate Python/PySpark code to invoke a data preparation package

You are ready to move on to the next part in the tutorial series, where you learn how to build an Azure Machine Learning model:

[Tutorial 2 - Build models](#)

Tutorial 2: Classify Iris - Build a model

12/11/2018 • 17 minutes to read • [Edit Online](#)

NOTE

This article is deprecated. Support for earlier versions of this service will end incrementally. [View the support timeline](#). Start using the latest version with this [quickstart](#).

Azure Machine Learning service (preview) are an integrated, data science and advanced analytics solution for professional data scientists to prepare data, develop experiments, and deploy models at cloud scale.

This tutorial is **part two of a three-part series**. In this part of the tutorial, you use Azure Machine Learning service to:

- Open scripts and review code
- Execute scripts in a local environment
- Review run histories
- Execute scripts in a local Azure CLI window
- Execute scripts in a local Docker environment
- Execute scripts in a remote Docker environment
- Execute scripts in a cloud Azure HDInsight environment

This tutorial uses the timeless [Iris flower data set](#).

Prerequisites

To complete this tutorial, you need:

- An Azure subscription. If you don't have an Azure subscription, create a [free account](#) before you begin.
- An experimentation account and Azure Machine Learning Workbench installed as described in this [quickstart](#)
- The project and prepared Iris data from [Tutorial part 1](#)
- A Docker engine installed and running locally. Docker's Community Edition is sufficient. Learn how to install Docker here: <https://docs.docker.com/engine/installation/>.

Review iris_sklearn.py and the configuration files

1. Launch the Azure Machine Learning Workbench application.
2. Open the **myIris** project you created in [Part 1 of the tutorial series](#).
3. In the open project, select the **Files** button (the folder icon) on the far-left pane to open the file list in your project folder.

Project Dashboard

I myIris

LOCATION /Users/admin/temp/myIris

local iris_sklearn.py Arguments: Arguments

Classifying Iris

This is a companion sample project of the Azure Machine Learning QuickStart and Tutorials. dataset, creating a model and deploying it as a web service.

4. Select the **iris_sklearn.py** Python script file.

IrisGarden

I myIris

LOCATION /Users/admin/temp/myIris

local iris_sklearn.py Arguments: Arguments

Classifying Iris

This is a companion sample project of the Azure Machine Learning QuickStart and Tutorials. dataset, creating a model and deploying it as a web service.

The code opens in a new text editor tab inside the Workbench. This is the script you use throughout this part of the tutorial.

NOTE

The code you see might not be exactly the same as the preceding code, because this sample project is updated frequently.

The screenshot shows the Azure Machine Learning Workbench interface. The left sidebar displays a file tree for a project named 'myIris'. The main area shows the content of the file 'iris_sklearn.py'. The code is a Python script that imports various libraries like pickle, sys, os, numpy, sklearn, and matplotlib, and performs tasks such as loading the Iris dataset, creating a logistic regression model, and plotting ROC curves.

```
1 # Please make sure scikit-learn is included the conda_dependencies.yml file.
2 import pickle
3 import sys
4 import os
5
6 import numpy as np
7 from sklearn.metrics import confusion_matrix
8
9 from sklearn.linear_model import LogisticRegression
10 from sklearn.model_selection import train_test_split
11 from sklearn.metrics import precision_recall_curve
12
13 from azureml.logging import get_azureml_logger
14 from azureml.dataprep.package import run
15
16 from plot_graphs import plot_iris
17
18 # initialize the logger
19 run_logger = get_azureml_logger()
20
21 # create the outputs folder
22 os.makedirs('./outputs', exist_ok=True)
23
24 print('Python version: {}'.format(sys.version))
25 print()
26
27 # load Iris dataset from a DataPrep package as a pandas DataFrame
28 iris = run('iris.dprep', dataflow_idx=0, spark=False)
29 print ('Iris dataset shape: {}'.format(iris.shape))
```

5. Inspect the Python script code to become familiar with the coding style.

The script **iris_sklearn.py** performs the following tasks:

- Loads the default data preparation package called **iris.dprep** to create a [pandas DataFrame](#).
- Adds random features to make the problem more difficult to solve. Randomness is necessary because Iris is a small data set that is easily classified with nearly 100% accuracy.
- Uses the [scikit-learn](#) machine learning library to build a logistic regression model. This library comes with Azure Machine Learning Workbench by default.
- Serializes the model using the [pickle](#) library into a file in the `outputs` folder.
- Loads the serialized model, and then deserializes it back into memory.
- Uses the deserialized model to make a prediction on a new record.
- Plots two graphs, a confusion matrix and a multi-class receiver operating characteristic (ROC) curve, using the [matplotlib](#) library, and then saves them in the `outputs` folder. You can install this library in your environment if it isn't there already.
- Plots the regularization rate and model accuracy in the run history automatically. The `run_logger` object is used throughout to record the regularization rate and the model accuracy into the logs.

Run `iris_sklearn.py` in your local environment

1. Start the Azure Machine Learning command-line interface (CLI):

- a. Launch the Azure Machine Learning Workbench.
- b. From the Workbench menu, select **File > Open Command Prompt**.

The Azure Machine Learning command-line interface (CLI) window starts in the project folder `C:\Temp\myIris\>` on Windows. This project is the same as the one you created in Part 1 of the tutorial.

IMPORTANT

You must use this CLI window to accomplish the next steps.

2. In the CLI window, install the Python plotting library, **matplotlib**, if you do not already have the library.

The **iris_sklearn.py** script has dependencies on two Python packages: **scikit-learn** and **matplotlib**. The **scikit-learn** package is installed by Azure Machine Learning Workbench for your convenience. But, you need to install **matplotlib** if you don't have it installed yet.

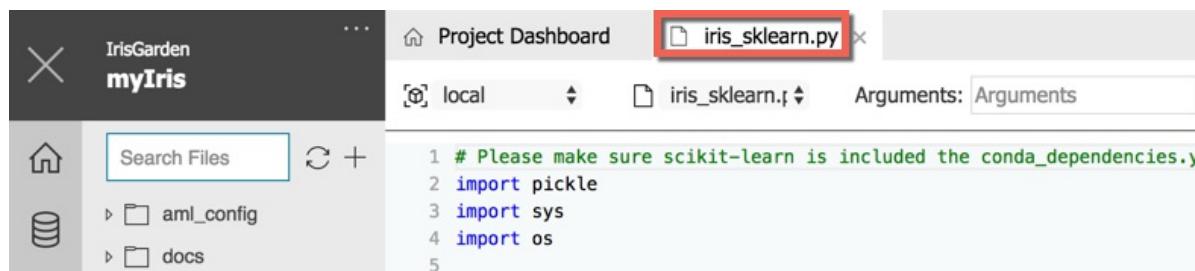
If you move on without installing **matplotlib**, the code in this tutorial can still run successfully. However, the code will not be able to produce the confusion matrix output and the multi-class ROC curve plots shown in the history visualizations.

```
pip install matplotlib
```

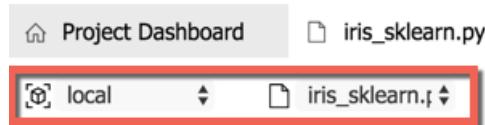
This install takes about a minute.

3. Return to the Workbench application.

4. Find the tab called **iris_sklearn.py**.



5. In the toolbar of that tab, select **local** as the execution environment, and **iris_sklearn.py** as the script to run. These may already be selected.

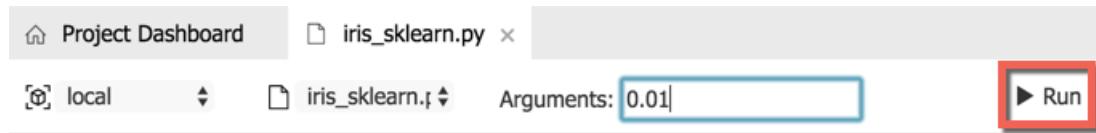


6. Move to the right side of the toolbar and enter **0.01** in the **Arguments** field.

This value corresponds to the regularization rate of the logistic regression model.



7. Select the **Run** button. A job is immediately scheduled. The job is listed in the **Jobs** pane on the right side of the workbench window.



After a few moments, the status of the job transitions from **Submitting**, to **Running**, and finally to **Completed**.

8. Select **Completed** in the job status text in the **Jobs** pane.

The screenshot shows the Project Dashboard interface. In the center, there's a code editor window titled "iris_sklearn.py" with the following content:

```

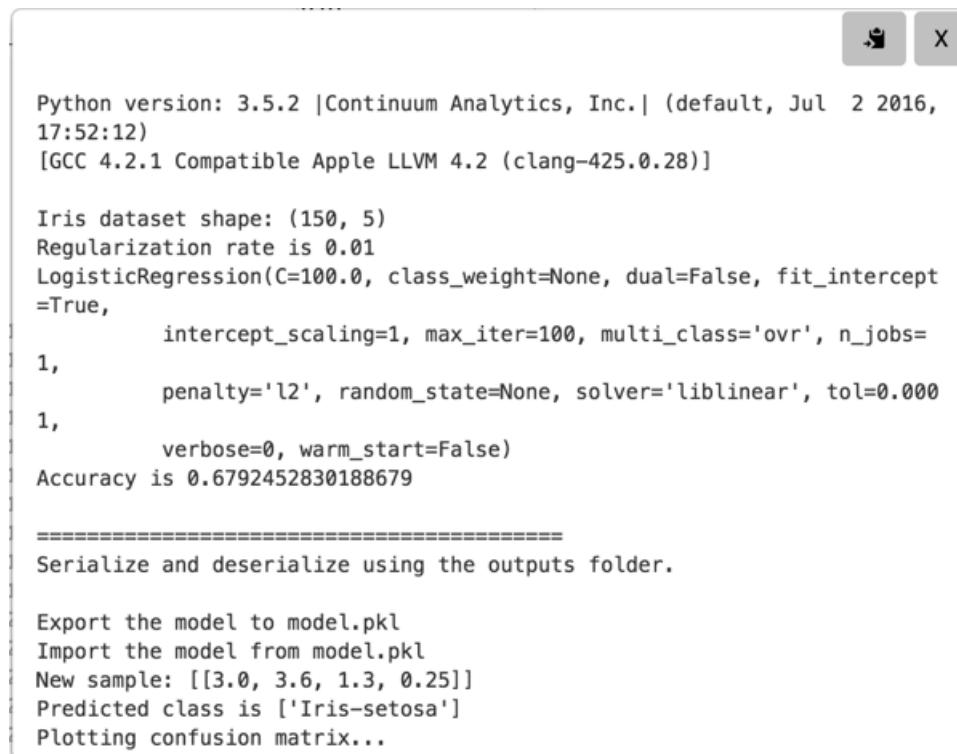
1 # Please make sure scikit-learn is included the conda_dependencies.yml file.
2 import pickle
3 import sys
4 import os
5
6 import numpy as np
7 from sklearn.metrics import confusion_matrix

```

Below the code editor, there are tabs for "local" and "iris_sklearn". The "Arguments" field contains "0.01". To the right, there are "Run", "Edit", and "Jobs" buttons. The "Jobs" pane on the right shows a single entry:

- Job name: "iris_sklearn.py [5]"
- Status: "Completed" (highlighted with a red box)
- Time: "Today at 9:04 PM"

A pop-up window opens and displays the standard output (stdout) text for the run. To close the stdout text, select the **Close (x)** button on the upper right of the pop-up window.



- In the same job status in the **Jobs** pane, select the blue text **iris_sklearn.py [n]** (*n* is the run number) just above the **Completed** status and the start time. The **Run Properties** window opens and shows the following information for that particular run:

- **Run Properties** information
- **Outputs**
- **Metrics**
- **Visualizations**, if any
- **Logs**

When the run is finished, the pop-up window shows the following results:

NOTE

Because the tutorial introduced some randomization into the training set earlier, your exact results might vary from the results shown here.

```

Python version: 3.5.2 |Continuum Analytics, Inc.| (default, Jul 5 2016, 11:41:13) [MSC v.1900 64 bit
(AMD64)]

Iris dataset shape: (150, 5)
Regularization rate is 0.01
LogisticRegression(C=100.0, class_weight=None, dual=False, fit_intercept=True,
    intercept_scaling=1, max_iter=100, multi_class='ovr', n_jobs=1,
    penalty='l2', random_state=None, solver='liblinear', tol=0.0001,
    verbose=0, warm_start=False)
Accuracy is 0.6792452830188679

=====
Serialize and deserialize using the outputs folder.

Export the model to model.pkl
Import the model from model.pkl
New sample: [[3.0, 3.6, 1.3, 0.25]]
Predicted class is ['Iris-setosa']
Plotting confusion matrix...
Confusion matrix in text:
[[50  0  0]
 [ 1 37 12]
 [ 0  4 46]]
Confusion matrix plotted.
Plotting ROC curve....
ROC curve plotted.
Confusion matrix and ROC curve plotted. See them in Run History details pane.

```

10. Close the **Run Properties** tab, and then return to the **iris_sklearn.py** tab.

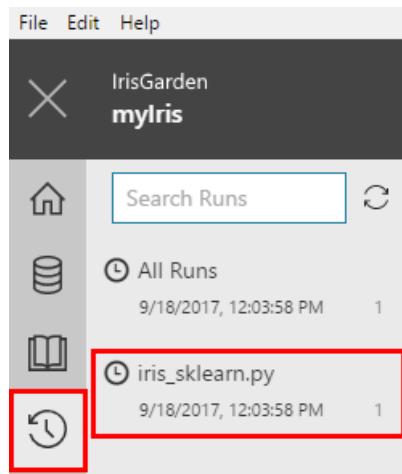
11. Repeat for additional runs.

Enter a series of values in the **Arguments** field ranging from `0.001` to `10`. Select **Run** to execute the code a few more times. The argument value you change each time is fed to the logistic regression model in the code, resulting in different findings each time.

Review the run history in detail

In Azure Machine Learning Workbench, every script execution is captured as a run history record. If you open the **Runs** view, you can view the run history of a particular script.

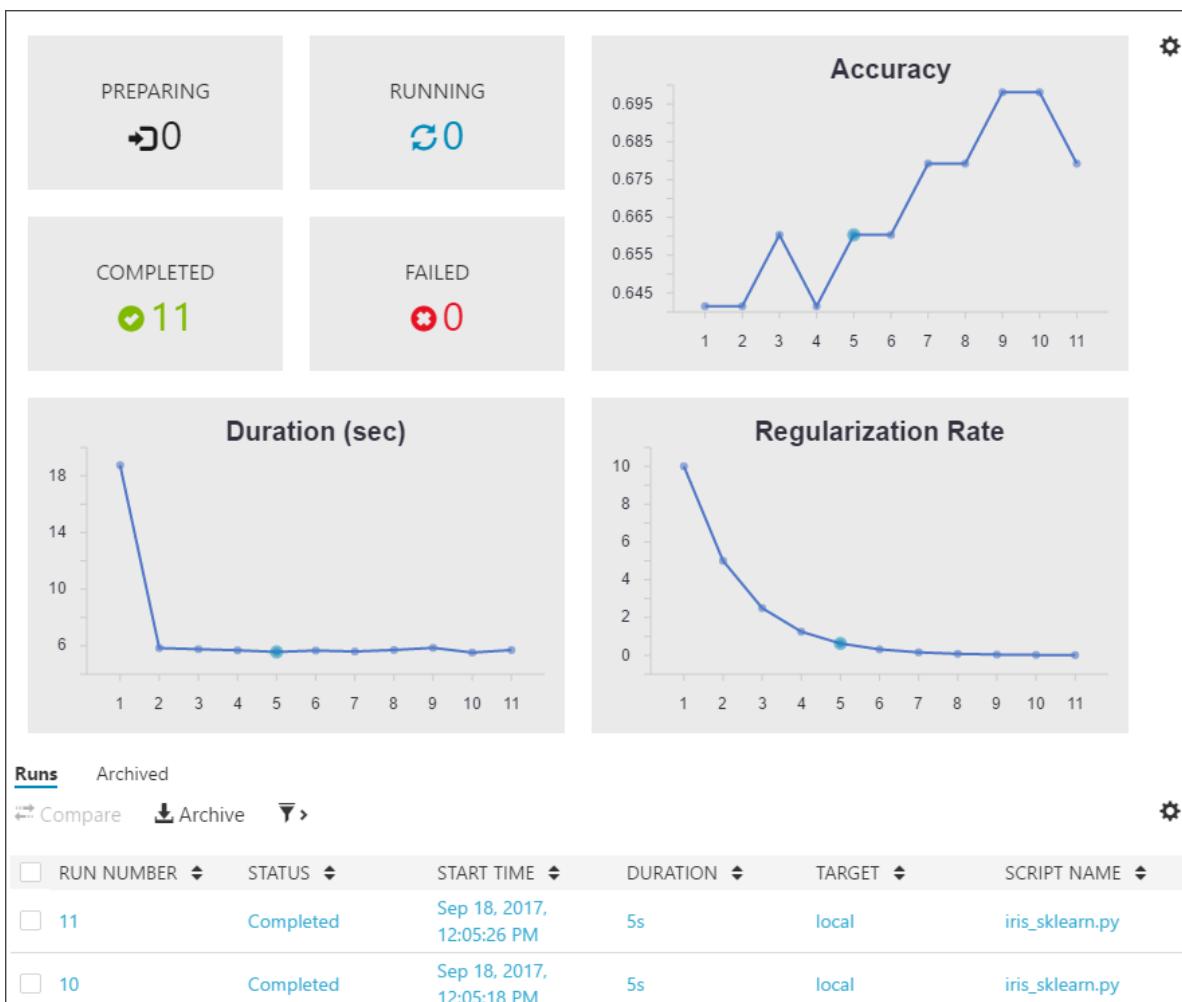
- To open the list of **Runs**, select the **Runs** button (clock icon) on the left toolbar. Then select **iris_sklearn.py** to show the **Run Dashboard** of `iris_sklearn.py`.



- The **Run Dashboard** tab opens.

Review the statistics captured across the multiple runs. Graphs render in the top of the tab. Each run has a

consecutive number, and the run details are listed in the table at the bottom of the screen.



3. Filter the table, and then select any of the graphs to view the status, duration, accuracy, and regularization rate of each run.
4. Select the checkboxes next to two or more runs in the **Runs** table. Select the **Compare** button to open a detailed comparison pane. Review the side-by-side comparison.
5. To return to the **Run Dashboard**, select the **Run List** back button on the upper left of the **Comparison** pane.

The screenshot shows the Run List pane with the following components:

- Header:** Project Dashboard, iris_sklearn.py, iris_sklearn.py
- Buttons:** ← Run List (highlighted with a red box), + Add Run
- Section:** Compare Runs
- Table:** A comparison table for runNumber 1 and 5. The columns are runNumber, experimentId, and value.

runNumber	experimentId	1	5
experimentId	a656c77f-a1d4-48de-a225-1f316e34999d		a656c77f-a1d4-48de-a225-1f316e34999d

6. Select an individual run to see the run detail view. Notice that the statistics for the selected run are listed in the **Run Properties** section. The files written into the output folder are listed in the **Outputs** section, and you can download the files from there.

Project Dashboard iris_sklearn.py

Run List Compare Restore

Run Properties

Status	Completed
Start Time	Sep 18, 2017, 12:05:26 PM
Duration	5s
Target	local
Run Id	myIris_1505736326221
Run Number	11
Script Name	iris_sklearn.py
Git Commit Hash	ca18889d0ad60d05b998399062032f0e5987c9f1
Regularization Rate	0.009765625
Accuracy	0.6792452830188679

Outputs

Promote Download

- model.pkl
- cm.png
- roc.png

Visualizations

outputs/cm.png

The confusion matrix shows the following counts:

True Species	Iris-setosa	Iris-versicolor	Iris-virginica
Iris-setosa	43	0	3
Iris-versicolor	0	44	2
Iris-virginica	3	0	44

outputs/roc.png

The multi-class ROC curve plot displays the performance of the model across three classes. The legend indicates:

- micro-average ROC curve (area = 0.91)
- macro-average ROC curve (area = 0.93)
- ROC curve of class Iris-setosa (area = 1.00)
- ROC curve of class Iris-versicolor (area = 0.82)
- ROC curve of class Iris-virginica (area = 0.96)

The two plots, the confusion matrix and the multi-class ROC curve, are rendered in the **Visualizations** section. All the log files can also be found in the **Logs** section.

Run scripts in local Docker environments

Optionally, you can experiment with running scripts against a local Docker container. You can configure additional execution environments, such as Docker, and run your script in those environments.

NOTE

To experiment with dispatching scripts to run in a Docker container in a remote Azure VM or an Azure HDInsight Spark cluster, you can follow the [instructions to create an Ubuntu-based Azure Data Science Virtual Machine or HDInsight cluster](#).

- If you have not yet done so, install and start Docker locally on your Windows or MacOS machine. For more information, see the Docker installation instructions at <https://docs.docker.com/install/>. Community edition is sufficient.

2. On the left pane, select the **Folder** icon to open the **Files** list for your project. Expand the `aml_config` folder.

3. There are several environments that are preconfigured: **docker-python**, **docker-spark**, and **local**.

Each environment has two files, such as `docker.compute` (for both **docker-python** and **docker-spark**) and `docker-python.runconfig`. Open each file to see that certain options are configurable in the text editor.

To clean up, select **Close (x)** on the tabs for any open text editors.

4. Run the **iris_sklearn.py** script by using the **docker-python** environment:

- On the left toolbar, select the **Clock** icon to open the **Runs** pane. Select **All Runs**.
- On the top of the **All Runs** tab, select **docker-python** as the targeted environment instead of the default **local**.
- Next, move to the right side and select **iris_sklearn.py** as the script to run.
- Leave the **Arguments** field blank because the script specifies a default value.
- Select the **Run** button.

5. Observe that a new job starts. It appears in the **Jobs** pane on the right side of the workbench window.

When you run against Docker for the first time, the job takes a few extra minutes to finish.

Behind the scenes, Azure Machine Learning Workbench builds a new Docker file. The new file references the base Docker image specified in the `docker.compute` file and the dependency Python packages specified in the `conda_dependencies.yml` file.

The Docker engine performs the following tasks:

- Downloads the base image from Azure.
- Installs the Python packages specified in the `conda_dependencies.yml` file.
- Starts a Docker container.
- Copies or references, depending on the run configuration, the local copy of the project folder.
- Executes the `iris_sklearn.py` script.

In the end, you should see the exact same results as you do when you target **local**.

6. Now, let's try Spark. The Docker base image contains a preinstalled and configured Spark instance that you can use to execute a PySpark script. Using this base image is an easy way to develop and test your Spark program, without having to spend time installing and configuring Spark yourself.

Open the `iris_spark.py` file. This script loads the `iris.csv` data file, and uses the logistic regression algorithm from the Spark Machine Learning library to classify the Iris data set. Now change the run environment to **docker-spark** and the script to **iris_spark.py**, and then run it again. This process takes a little longer because a Spark session has to be created and started inside the Docker container. You can also see the stdout is different than the stdout of `iris_spark.py`.

7. Start a few more runs and play with different arguments.

8. Open the `iris_spark.py` file to see the logistic regression model built using the Spark Machine Learning library.

9. Interact with the **Jobs** pane, run a history list view, and run a details view of your runs across different execution environments.

Run scripts in the CLI window

1. Start the Azure Machine Learning command-line interface (CLI):

- Launch the Azure Machine Learning Workbench.
- From the Workbench menu, select **File > Open Command Prompt**.

The CLI prompt starts in the project folder `C:\Temp\myIris\>` on Windows. This is the project you created in Part 1 of the tutorial.

IMPORTANT

You must use this CLI window to accomplish the next steps.

2. In the CLI window, log in to Azure. [Learn more about az login](#).

You may already be logged in. In that case, you can skip this step.

- At the command prompt, enter:

```
az login
```

This command returns a code to use in your browser at <https://aka.ms/devicelogin>.

- Go to <https://aka.ms/devicelogin> in your browser.
- When prompted, enter the code, which you received in the CLI, into your browser.

The Workbench app and CLI use independent credential caches when authenticating against Azure resources. After you log in, you won't need to authenticate again until the cached token expires.

3. If your organization has multiple Azure subscriptions (enterprise environment), you must set the subscription to be used. Find your subscription, set it using the subscription ID, and then test it.

- List every Azure subscription to which you have access using this command:

```
az account list -o table
```

The **az account list** command returns the list of subscriptions available to your login. If there is more than one, identify the subscription ID value for the subscription you want to use.

- Set the Azure subscription you want to use as the default account:

```
az account set -s <your-subscription-id>
```

where `<your-subscription-id>` is ID value for the subscription you want to use. Do not include the brackets.

- Confirm the new subscription setting by requesting the details for the current subscription.

```
az account show
```

4. In the CLI window, install the Python plotting library, **matplotlib**, if you do not already have the library.

```
pip install matplotlib
```

5. In the CLI window, submit the **iris_sklearn.py** script as an experiment.

The execution of `iris_sklearn.py` is run against the local compute context.

- On Windows:

```
az ml experiment submit -c local .\iris_sklearn.py
```

- On MacOS:

```
az ml experiment submit -c local iris_sklearn.py
```

Your output should be similar to:

```
RunId: myIris_1521077190506

Executing user inputs .....
=====

Python version: 3.5.2 |Continuum Analytics, Inc.| (default, Jul 2 2016, 17:52:12)
[GCC 4.2.1 Compatible Apple LLVM 4.2 (clang-425.0.28)]

Iris dataset shape: (150, 5)
Regularization rate is 0.01
LogisticRegression(C=100.0, class_weight=None, dual=False, fit_intercept=True,
    intercept_scaling=1, max_iter=100, multi_class='ovr', n_jobs=1,
    penalty='l2', random_state=None, solver='liblinear', tol=0.0001,
    verbose=0, warm_start=False)
Accuracy is 0.6792452830188679

=====
Serialize and deserialize using the outputs folder.

Export the model to model.pkl
Import the model from model.pkl
New sample: [[3.0, 3.6, 1.3, 0.25]]
Predicted class is ['Iris-setosa']
Plotting confusion matrix...
Confusion matrix in text:
[[50  0  0]
 [ 1 37 12]
 [ 0  4 46]]
Confusion matrix plotted.
Plotting ROC curve....
ROC curve plotted.
Confusion matrix and ROC curve plotted. See them in Run History details page.

Execution Details
=====
RunId: myIris_1521077190506
```

6. Review the output. You have the same output and results that you had when you used the Workbench to run the script.
7. In the CLI window, run the Python script, **iris_sklearn.py**, again using a Docker execution environment (if you have Docker installed on your machine).
 - If your container is on Windows:

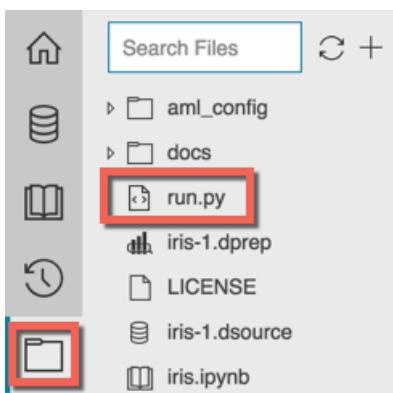
EXECUTION ENVIRONMENT	COMMAND ON WINDOWS
Python	<code>az ml experiment submit -c docker-python .\\iris_sklearn.py 0.01</code>
Spark	<code>az ml experiment submit -c docker-spark .\\iris_spark.py 0.1</code>

- If your container is on MacOS:

EXECUTION ENVIRONMENT	COMMAND ON WINDOWS
Python	<code>az ml experiment submit -c docker-python iris_sklearn.py 0.01</code>
Spark	<code>az ml experiment submit -c docker-spark iris_spark.py 0.1</code>

8. Go back to the Workbench, and:

- Select the folder icon on the left pane to list the project files.
- Open the Python script named **run.py**. This script is useful to loop over various regularization rates.



9. Run the experiment multiple times with those rates.

This script starts an `iris_sklearn.py` job with a regularization rate of `10.0` (a ridiculously large number). The script then cuts the rate to half in the following run, and so on, until the rate is no smaller than `0.005`.

The script contains the following code:

```

1 # run iris_sklearn.py with descending regularization rates
2 # run this with just "python run.py". It will fail if you run using az ml execute.
3
4 import os
5
6 # set regularization rate as an argument
7 reg = 10
8 while reg > 0.005:
9     os.system('az ml experiment submit -c local ./iris_sklearn.py {}'.format(reg))
10    # cut regularization rate to half
11    reg = reg / 2

```

10. Run the **run.py** script from the command line as follows:

```
python run.py
```

This command submits `iris_sklearn.py` multiple times with different regularization rates

When `run.py` finishes, you can see graphs of different metrics in your run history list view in the workbench.

Run scripts in a remote Docker container

To execute your script in a Docker container on a remote Linux machine, you need to have SSH access (username and password) to that remote machine. In addition, the machine must have a Docker engine installed and running. The easiest way to obtain such a Linux machine is to create an Ubuntu-based Data Science Virtual Machine (DSVM) on Azure. Learn [how to create an Ubuntu DSVM to use in Azure ML Workbench](#).

NOTE

The CentOS-based DSVM is *not* supported.

1. After the VM is created, you can attach the VM as an execution environment by generating a pair of `.runconfig` and `.compute` files. Use the following command to generate the files.

Let's name the new compute target `myvm`.

```
az ml computetarget attach remotemachine --name myvm --address <your-IP> --username <your-username> --password <your-password>
```

NOTE

The IP address can also be a publicly addressable fully-qualified domain name (FQDN) such as `vm-name.southcentralus.cloudapp.azure.com`. It is a good practice to add an FQDN to your DSVM and use it instead of an IP address. This practice is a good idea because you might turn off the VM at some point to save on cost. Additionally, the next time you start the VM, the IP address might have changed.

NOTE

In addition to username and password authentication, you can specify a private key and the corresponding passphrase (if any) using the `--private-key-file` and (optionally) the `--private-key-passphrase` options. If you want to use the private key that you used when created DSVM, you should specify the `--use-azureml-ssh-key` option.

Next, prepare the `myvm` compute target by running this command.

```
az ml experiment prepare -c myvm
```

The preceding command constructs the Docker image in the VM to get it ready to run the scripts.

NOTE

You can also change the value of `PrepareEnvironment` in `myvm.runconfig` from the default value `false` to `true`. This change automatically prepares the Docker container as part of the first run.

2. Edit the generated `myvm.runconfig` file under `aml_config` and change the framework from the default value `PySpark` to `Python`:

Framework: Python

NOTE

While PySpark should also work, using Python is more efficient if you don't actually need a Spark session to run your Python script.

3. Issue the same command as you did before in the CLI window, using target `myvm` this time to execute `iris_sklearn.py` in a remote Docker container:

```
az ml experiment submit -c myvm iris_sklearn.py
```

The command executes as if you're in a `docker-python` environment, except that the execution happens on the remote Linux VM. The CLI window displays the same output information.

4. Let's try using Spark in the container. Open File Explorer. Make a copy of the `myvm.runconfig` file and name it `myvm-spark.runconfig`. Edit the new file to change the `Framework` setting from `Python` to `PySpark`:

Framework: PySpark

Don't make any changes to the `myvm.compute` file. The same Docker image on the same VM gets used for the Spark execution. In the new `myvm-spark.runconfig`, the `Target` field points to the same `myvm.compute` file via its name `myvm`.

5. Type the following command to run the `iris_spark.py` script in the Spark instance running inside the remote Docker container:

```
az ml experiment submit -c myvm-spark .\iris_spark.py
```

Run scripts in HDInsight clusters

You can also run this script in an HDInsight Spark cluster. Learn [how to create an HDInsight Spark Cluster to use in Azure ML Workbench](#).

NOTE

The HDInsight cluster must use Azure Blob as the primary storage. Using Azure Data Lake storage is not supported yet.

1. If you have access to a Spark for Azure HDInsight cluster, generate an HDInsight run configuration command as shown here. Provide the HDInsight cluster name and your HDInsight username and password as the parameters.

Use the following command to create a compute target that points to a HDInsight cluster:

```
az ml computetarget attach cluster --name myhdi --address <cluster head node FQDN> --username <your-username> --password <your-password>
```

To prepare the HDInsight cluster, run this command:

```
az ml experiment prepare -c myhdi
```

The cluster head node FQDN is typically <your_cluster_name>-ssh.azurehdinsight.net.

NOTE

The `username` is the cluster SSH username defined during HDInsight setup. By default, the value is `sshuser`. The value is not `admin`, which is the other user created during setup to enable access to the cluster's admin website.

- Run the **iris_spark.py** script in the HDInsight cluster with this command:

```
az ml experiment submit -c myhdi .\iris_spark.py
```

NOTE

When you execute against a remote HDInsight cluster, you can also view the Yet Another Resource Negotiator (YARN) job execution details at https://<your_cluster_name>.azurehdinsight.net/yarnui by using the `admin` user account.

Clean up resources

IMPORTANT

The resources you created can be used as prerequisites to other Azure Machine Learning service tutorials and how-to articles.

If you don't plan to use the resources you created here, delete them so you don't incur any charges.

- In the Azure portal, select **Resource groups** on the far left.

The screenshot shows the Microsoft Azure portal interface. The left sidebar has a 'Resource groups' item highlighted with a red box. The main area shows the 'newacct' resource group details. At the top right of the main area, there is a 'Delete resource group' button, which is also highlighted with a red box.

- From the list, select the resource group you created.
- Select **Delete resource group**.
- Enter the resource group name, and then select **Delete**.

Next steps

In this second part of the three-part tutorial series, you learned how to:

- Open scripts and review the code in Workbench
- Execute scripts in a local environment
- Review the run history
- Execute scripts in a local Docker environment

Now, you can try out the third part of this tutorial series in which you can deploy the logistic regression model you created as a real-time web service.

[Tutorial 3 - Deploy models](#)

Tutorial 3: Classify Iris: Deploy a model

12/11/2018 • 13 minutes to read • [Edit Online](#)

NOTE

This article is deprecated. Support for earlier versions of this service will end incrementally. [View the support timeline.](#)
Start using the latest version with this [quickstart](#).

Azure Machine Learning (preview) is an integrated, end-to-end data science and advanced analytics solution for professional data scientists. Data scientists can use it to prepare data, develop experiments, and deploy models at cloud scale.

This tutorial is **part three of a three-part series**. In this part of the tutorial, you use Machine Learning (preview) to:

- Locate the model file.
- Generate a scoring script and schema file.
- Prepare the environment.
- Create a real-time web service.
- Run the real-time web service.
- Examine the output blob data.

This tutorial uses the timeless [Iris flower data set](#).

Prerequisites

To complete this tutorial, you need:

- An Azure subscription. If you don't have an Azure subscription, create a [free account](#) before you begin.
- An experimentation account and Azure Machine Learning Workbench installed as described in this [quickstart](#)
- The classification model from [Tutorial part 2](#)
- A Docker engine installed and running locally

Download the model pickle file

In the previous part of the tutorial, the **iris_sklearn.py** script was run in the Machine Learning Workbench locally. This action serialized the logistic regression model by using the popular Python object-serialization package [pickle](#).

1. Open the Machine Learning Workbench application. Then open the **myIris** project you created in the previous parts of the tutorial series.
2. After the project is open, select the **Files** button (folder icon) on the left pane to open the file list in your project folder.
3. Select the **iris_sklearn.py** file. The Python code opens in a new text editor tab inside the workbench.
4. Review the **iris_sklearn.py** file to see where the pickle file was generated. Select Ctrl+F to open the **Find** dialog box, and then find the word **pickle** in the Python code.

This code snippet shows how the pickle output file was generated. The output pickle file is named **model.pkl** on the disk.

```

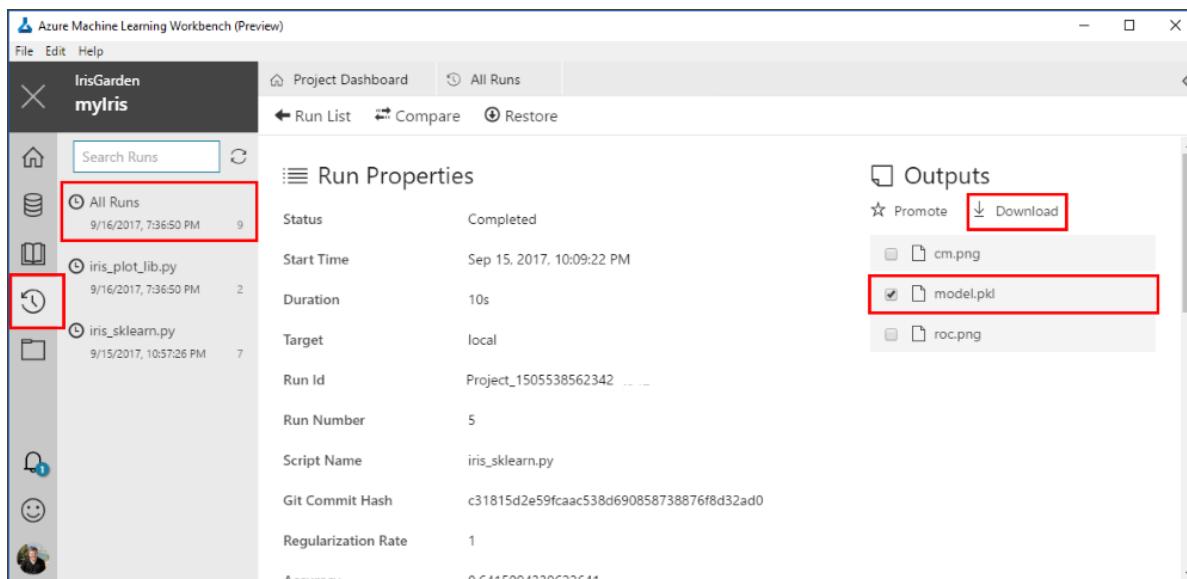
print("Export the model to model.pkl")
f = open('./outputs/model.pkl', 'wb')
pickle.dump(clf1, f)
f.close()

```

- Locate the model pickle file in the output files of a previous run.

When you ran the **iris_sklearn.py** script, the model file was written to the **outputs** folder with the name **model.pkl**. This folder lives in the execution environment that you choose to run the script, and not in your local project folder.

- To locate the file, select the **Runs** button (clock icon) on the left pane to open the list of **All Runs**.
- The **All Runs** tab opens. In the table of runs, select one of the recent runs where the target was **local** and the script name was **iris_sklearn.py**.
- The **Run Properties** pane opens. In the upper-right section of the pane, notice the **Outputs** section.
- To download the pickle file, select the check box next to the **model.pkl** file, and then select **Download**. Save the file to the root of your project folder. The file is needed in the upcoming steps.



Read more about the `outputs` folder in [How to read and write large data files](#).

Get the scoring script and schema files

To deploy the web service along with the model file, you also need a scoring script. Optionally, you need a schema for the web service input data. The scoring script loads the **model.pkl** file from the current folder and uses it to produce new predictions.

- Open the Machine Learning Workbench application. Then open the **myIris** project you created in the previous part of the tutorial series.
- After the project is open, select the **Files** button (folder icon) on the left pane to open the file list in your project folder.
- Select the **score_iris.py** file. The Python script opens. This file is used as the scoring file.

```

1 # This script generates the scoring and schema files
2 # Creates the schema, and holds the init and run functions needed to
3 # operationalize the Iris Classification sample
4
5 # Import data collection library. Only supported for docker mode.
6 # Functionality will be ignored when package isn't found
7 try:
8     from azureml.datacollector import ModelDataCollector
9 except ImportError:
10    print("Data collection is currently only supported in docker mode. May be disabled fo
11    # Mocking out model data collector functionality
12    class ModelDataCollector(object):
13        def __init__(self, *args, **kw): pass
14        def __getattribute__(self, _): return self.__init__
15        def __new__(self, *args, **kw): return None
16    pass
17
18 import os
19
20 # Prepare the web service definition by authoring
21 # init() and run() functions. Test the functions
22 # before deploying the web service.
23 def init():
24     global inputs_dc, prediction_dc
25     from sklearn.externals import joblib
26
27     # load the model file
28     global model

```

- To get the schema file, run the script. Select the **local** environment and the **score_iris.py** script in the command bar, and then select **Run**.

This script creates a JSON file in the **Outputs** section, which captures the input data schema required by the model.

- Note the **Jobs** pane on the right side of the **Project Dashboard** pane. Wait for the latest **score_iris.py** job to display the green **Completed** status. Then select the hyperlink **score_iris.py** for the latest job run to see the run details.
- On the **Run Properties** pane, in the **Outputs** section, select the newly created **service_schema.json** file. Select the check box next to the file name, and then select **Download**. Save the file into your project root folder.
- Return to the previous tab where you opened the **score_iris.py** script. By using data collection, you can capture model inputs and predictions from the web service. The following steps are of particular interest for data collection.
- Review the code at the top of the file, which imports class **ModelDataCollector**, because it contains the model data collection functionality:

```
from azureml.datacollector import ModelDataCollector
```

- Review the following lines of code in the **init()** function that instantiates **ModelDataCollector**:

```
global inputs_dc, prediction_dc
inputs_dc = ModelDataCollector('model.pkl', identifier="inputs")
prediction_dc = ModelDataCollector('model.pkl', identifier="prediction")`
```

- Review the following lines of code in the **run(input_df)** function as it collects the input and prediction data:

```
inputs_dc.collect(input_df)
prediction_dc.collect(pred)
```

Now you're ready to prepare your environment to operationalize the model.

Prepare to operationalize locally [For development and testing your service]

Use *local mode* deployment to run in Docker containers on your local computer.

You can use *local mode* for development and testing. The Docker engine must be running locally to complete the following steps to operationalize the model. You can use the `-h` flag at the end of each command to show the corresponding help message.

NOTE

If you don't have the Docker engine locally, you can still proceed by creating a cluster in Azure for deployment. You can keep the cluster for re-use, or delete it after the tutorial so you don't incur ongoing charges.

NOTE

Web services deployed locally do not show up in Azure Portal's list of services. They will be running in Docker on the local machine.

1. Open the command-line interface (CLI). In the Machine Learning Workbench application, on the **File** menu, select **Open Command Prompt**.

The command-line prompt opens in your current project folder location `c:\temp\myIris>`.

2. Make sure the Azure resource provider **Microsoft.ContainerRegistry** is registered in your subscription.

You must register this resource provider before you can create an environment in step 3. You can check to see if it's already registered by using the following command:

```
az provider list --query "[].{Provider:namespace, Status:registrationState}" --out table
```

You should see output like this:

Provider	Status
-----	-----
Microsoft.Authorization	Registered
Microsoft.ContainerRegistry	Registered
microsoft.insights	Registered
Microsoft.MachineLearningExperimentation	Registered
...	

If **Microsoft.ContainerRegistry** is not registered, you can register it by using the following command:

```
az provider register --namespace Microsoft.ContainerRegistry
```

Registration can take a few minutes. You can check on its status by using the previous **az provider list** command or the following command:

```
az provider show -n Microsoft.ContainerRegistry
```

The third line of the output displays "**registrationState": "Registering"**". Wait a few moments and repeat the **show** command until the output displays "**registrationState": "Registered"**".

NOTE

If you are deploying to an ACS cluster, you need register the **Microsoft.ContainerService** resource provider as well using the exact same approach.

3. Create the environment. You must run this step once per environment. For example, run it once for development environment, and once for production. Use *local mode* for this first environment. You can try the `-c` or `--cluster` switch in the following command to set up an environment in *cluster mode* later.

The following setup command requires you to have Contributor access to the subscription. If you don't have that, you need at least Contributor access to the resource group that you are deploying to. In the latter case, you need to specify the resource group name as part of the setup command by using the `-g` flag.

```
az ml env setup -n <new deployment environment name> --location <e.g. eastus2>
```

Follow the on-screen instructions to provision a storage account for storing Docker images, an Azure container registry that lists the Docker images, and an Azure Application Insights account that gathers telemetry. If you use the `-c` switch, the command will additionally create a Container Service cluster.

The cluster name is a way for you to identify the environment. The location should be the same as the location of the Model Management account you created from the Azure portal.

To make sure that the environment is set up successfully, use the following command to check the status:

```
az ml env show -n <deployment environment name> -g <existing resource group name>
```

Make sure that "Provisioning State" has the value "Succeeded", as shown, before you set the environment in step 5:

```
C:\Work\ML\Vienna\Projects\IrisDemo>az ml env show -g iris2018rg -n iris2018
{
  "Cluster Name": "iris2018",
  "Cluster Size": "N/A",
  "Created On": "2018-01-25T18:47:17.921999999999999Z",
  "Location": "eastus2"
  "Provisioning State": "Succeeded",
  "Resource Group": "iris2018rg",
  "Subscription": ""
}
```

4. If you didn't create a Model Management account in previous parts of this tutorial, do so now. This is a one-time setup.

```
az ml account modelmanagement create --location <e.g. eastus2> -n <new model management account name> -g <existing resource group name> --sku-name S1
```

5. Set the Model Management account.

```
az ml account modelmanagement set -n <youracctname> -g <yourresourcegroupname>
```

6. Set the environment.

After the setup finishes, use the following command to set the environment variables required to operationalize the environment. Use the same environment name that you used previously in step 3. Use the same resource group name that was output in the command window when the setup process finished.

```
az ml env set -n <deployment environment name> -g <existing resource group name>
```

7. To verify that you have properly configured your operationalized environment for local web service deployment, enter the following command:

```
az ml env show
```

Now you're ready to create the real-time web service.

NOTE

You can reuse your Model Management account and environment for subsequent web service deployments. You don't need to create them for each web service. An account or an environment can have multiple web services associated with it.

Create a real-time web service in one command

1. To create a real-time web service, use the following command:

```
az ml service create realtime -f score_iris.py --model-file model.pkl -s ./output/service_schema.json -n irisapp -r python --collect-model-data true -c aml_config\conda_dependencies.yml
```

This command generates a web service ID you can use later. Omit the output directory if in a notebook.

The following switches are used with the **az ml service create realtime** command:

- **-f** : The scoring script file name.
- **--model-file** : The model file. In this case, it's the pickled model.pkl file.
- **-s** : The service schema. This was generated in a previous step by running the **score_iris.py** script locally.
- **-n** : The app name, which must be all lowercase.
- **-r** : The runtime of the model. In this case, it's a Python model. Valid runtimes are **python** and **spark-py**.
- **--collect-model-data true** : This switch enables data collection.
- **-c** : Path to the conda dependencies file where additional packages are specified.

IMPORTANT

The service name, which is also the new Docker image name, must be all lowercase. Otherwise, you get an error.

2. When you run the command, the model and the scoring files are uploaded to the storage account you created as part of the environment setup. The deployment process builds a Docker image with your model, schema, and scoring file in it, and then pushes it to the Azure container registry:

<ACR_name>.azurecr.io/<imagename>:<version>.

The command pulls down the image locally to your computer and then starts a Docker container based on that image. If your environment is configured in cluster mode, the Docker container is deployed into the Azure Cloud Services Kubernetes cluster instead.

As part of the deployment, an HTTP REST endpoint for the web service is created on your local machine. After a few minutes, the command should finish with a success message. Your web service is ready for action!

3. To see the running Docker container, use the **docker ps** command:

```
docker ps
```

[Optional alternative] Create a real-time web service by using separate commands

As an alternative to the **az ml service create realtime** command shown previously, you also can perform the steps separately.

First, register the model. Then generate the manifest, build the Docker image, and create the web service. This step-by-step approach gives you more flexibility at each step. Additionally, you can reuse the entities generated in previous steps and rebuild the entities only when needed.

1. Register the model by providing the pickle file name.

```
az ml model register --model model.pkl --name model.pkl
```

This command generates a model ID.

2. Create a manifest.

To create a manifest, use the following command and provide the model ID output from the previous step:

```
az ml manifest create --manifest-name <new manifest name> -f score_iris.py -r python -i <model ID> -s ./output/service_schema.json -c aml_config\conda_dependencies.yml
```

This command generates a manifest ID. Omit the output directory if in a notebook.

3. Create a Docker image.

To create a Docker image, use the following command and provide the manifest ID value output from the previous step. You also can optionally include the conda dependencies by using the **-c** switch.

```
az ml image create -n irisimage --manifest-id <manifest ID>
```

This command generates a Docker image ID.

4. Create the service.

To create a service, use the following command and provide the image ID output from the previous step:

```
az ml service create realtime --image-id <image ID> -n irisapp --collect-model-data true
```

This command generates a web service ID.

You are now ready to run the web service.

Run the real-time web service

To test the **irisapp** web service that's running, use a JSON-encoded record containing an array of four random numbers.

1. The web service includes sample data. When running in local mode, you can call the **az ml service usage realtime** command. That call retrieves a sample run command that you can use to test the service. The call also retrieves the scoring URL that you can use to incorporate the service into your own custom app.

```
az ml service usage realtime -i <web service ID>
```

2. To test the service, execute the returned service run command:

```
az ml service run realtime -i <web service ID> -d '{"input_df": [{"petal width": 0.25, "sepal length": 3.0, "sepal width": 3.6, "petal length": 1.3}]}'
```

The output is "**Iris-setosa**", which is the predicted class. (Your result might be different.)

View the collected data in Azure Blob storage

1. Sign in to the [Azure portal](#).
2. Locate your storage accounts. To do so, select **All Services**.
3. In the search box, enter **Storage accounts**, and then select Enter.
4. From the **Storage accounts** search box, select the **Storage account** resource matching your environment.

TIP

To determine which storage account is in use:

1. Open Machine Learning Workbench.
2. Select the project you're working on.
3. Open a command line prompt from the **File** menu.
4. At the command line prompt, enter `az ml env show -v`, and check the *storage_account* value. This is the name of your storage account.

5. After the **Storage account** pane opens, select **Blobs** from the **Services** section. Locate the container named **modeldata**.

If you don't see any data, you might need to wait up to 10 minutes after the first web-service request to see the data propagate to the storage account.

Data flows into blobs with the following container path:

```
/modeldata/<subscription_id>/<resource_group_name>/<model_management_account_name>/<webservice_name>/<model_id>-<model_name>-<model_version>/<identifier>/<year>/<month>/<day>/data.csv
```

6. You can consume this data from Azure Blob storage. There are a variety of tools that use both Microsoft software and open-source tools, such as:

- Machine Learning: Open the CSV file by adding the CSV file as a data source.
- Excel: Open the daily CSV files as a spreadsheet.
- **Power BI**: Create charts with data pulled from the CSV data in blobs.
- **Hive**: Load the CSV data into a Hive table, and perform SQL queries directly against the blobs.
- **Spark**: Create a DataFrame with a large portion of CSV data.

```
var df =
spark.read.format("com.databricks.spark.csv").option("inferSchema","true").option("header","true")
.load("wasb://modeldata@<storageaccount>.blob.core.windows.net/<subscription_id>/<resource_group_name>/<model_management_account_name>/<webservice_name>/<model_id>-<model_name>-<model_version>/<identifier>/<year>/<month>/<date>/*")
```

Clean up resources

IMPORTANT

The resources you created can be used as prerequisites to other Azure Machine Learning service tutorials and how-to articles.

If you don't plan to use the resources you created here, delete them so you don't incur any charges.

1. In the Azure portal, select **Resource groups** on the far left.

The screenshot shows the Microsoft Azure portal interface. The left sidebar has a 'Resource groups' item selected, which is highlighted with a red box. The main content area shows the 'newacct' resource group details. At the top right of the content area, there is a 'Delete resource group' button, also highlighted with a red box. Below it, there's an 'Essentials' section and a table listing resources under '1 items'.

NAME	TYPE	LOCATION
newacct	Azure Cosmos DB account	South Central US

2. From the list, select the resource group you created.
3. Select **Delete resource group**.
4. Enter the resource group name, and then select **Delete**.

Next steps

In this third part of the three-part tutorial series, you have learned how to use Machine Learning to:

- Locate the model file.
- Generate a scoring script and schema file.
- Prepare the environment.
- Create a real-time web service.
- Run the real-time web service.

- Examine the output blob data.

You have successfully run a training script in various compute environments. You have also created, serialized, and operationalized a model through a Docker-based web service.

You are now ready to do advanced data preparation:

[Tutorial 4 - Advanced data preparation](#)

Tutorial: Use Azure Machine Learning Workbench for advanced data preparation (Bike share data)

9/24/2018 • 24 minutes to read • [Edit Online](#)

NOTE

This article is deprecated. Support for earlier versions of this service will end incrementally. [View the support timeline](#). Start using the latest version with this [quickstart](#).

Azure Machine Learning (preview) is an integrated, end-to-end data science and advanced analytics solution for professional data scientists to prepare data, develop experiments, and deploy models at cloud scale.

In this tutorial, you use Machine Learning (preview) to learn how to:

- Prepare data interactively with the Machine Learning data preparation tool.
- Import, transform, and create a test dataset.
- Generate a data preparation package.
- Run the data preparation package by using Python.
- Generate a training dataset by reusing the data preparation package for additional input files.
- Execute scripts in a local Azure CLI window.
- Execute scripts in a cloud Azure HDInsight environment.

If you don't have an Azure subscription, create a [free account](#) before you begin.

Prerequisites

- A local installation of Azure Machine Learning Workbench. For more information, follow the [installation quickstart](#).
- If you don't have the Azure CLI installed, follow the instructions to [install the latest Azure CLI version](#).
- An [HDInsights Spark cluster](#) created in Azure.
- An Azure storage account.
- Familiarity with how to create a new project in Workbench.
- Although it's not required, it's helpful to have [Azure Storage Explorer](#) installed so that you can upload, download, and view the blobs in your storage account.

Data acquisition

This tutorial uses the [Boston hubway dataset](#) and Boston weather data from [NOAA](#).

1. Download the data files from the following links to your local development environment:

- [Boston weather data](#)
- Hubway trip data from the hubway website:
 - [201501-hubway-tripdata.zip](#)
 - [201504-hubway-tripdata.zip](#)
 - [201510-hubway-tripdata.zip](#)
 - [201601-hubway-tripdata.zip](#)

- o [201604-hubway-tripdata.zip](#)
 - o [201610-hubway-tripdata.zip](#)
 - o [201701-hubway-tripdata.zip](#)
2. Unzip each .zip file after download.

Upload data files to Azure Blob storage

You can use Azure Blob storage to host your data files.

1. Use the same storage account that is used for the HDInsight cluster you use.

The screenshot shows the Azure portal interface for managing storage accounts. In the top navigation bar, the path 'Home > ml-learning > bikeshareml - Storage accounts' is visible. The main content area displays a table with one row for 'bikesharestorage'. The 'STORAGE' column shows the storage account name, the 'CONTAINER / DIRECTORY' column shows 'bikeshareml-def', and the 'DEFAULT' column has a checked checkbox and a copy icon. On the left side, there's a vertical sidebar with various links like 'SSH + Cluster login', 'HDInsight partner', 'External metastores', etc., and a 'Properties' section with a 'Storage accounts' link, which is also highlighted with a red box. Below the sidebar, there are sections for 'SUPPORT + TROUBLESHOOTING' and 'New support request'.

2. Create a new container named **data-files** to store the **BikeShare** data files.

3. Upload the data files. Upload `BostonWeather.csv` to a folder named `weather`. Upload the trip data files to a folder named `tripdata`.

The screenshot shows the Azure portal interface for uploading blobs. On the left, the 'Blob service' blade is open, showing the 'bikesharestorage' account and the 'data-files' container. The 'data-files' container has a list of blobs: 'bikeshareml-def', 'bootdiagnostics-bikeshare-72941...', and 'data-files'. On the right, a 'Upload blob' dialog is open. It shows the file 'BostonWeather.csv' selected in the 'Files' input field, with a red box around it. There are checkboxes for 'Overwrite if files already exist' and 'Upload .vhdx files as page blobs (recommended)'. The 'Upload to folder' field contains 'weather', with a red box around it. At the bottom is a large blue 'Upload' button.

TIP

You also can use Storage Explorer to upload blobs. Use this tool when you want to view the contents of any files generated in the tutorial, too.

Learn about the datasets

1. The **Boston weather** file contains the following weather-related fields, reported on an hourly basis:

- **DATE**
- **REPORTTPYE**
- **HOURLYDRYBULBTEMPF**
- **HOURLYRelativeHumidity**
- **HOURLYWindSpeed**

2. The **hubway** data is organized into files by year and month. For example, the file named

`201501-hubway-tripdata.zip` contains a .csv file that contains data for January 2015. The data contains the following fields, with each row representing a bike trip:

- **Trip Duration (in seconds)**
- **Start Time and Date**
- **Stop Time and Date**
- **Start Station Name & ID**
- **End Station Name & ID**
- **Bike ID**
- **User Type (Casual = 24-Hour or 72-Hour Pass user; Member = Annual or Monthly Member)**
- **ZIP Code (if user is a member)**
- **Gender (self-reported by member)**

Create a new project

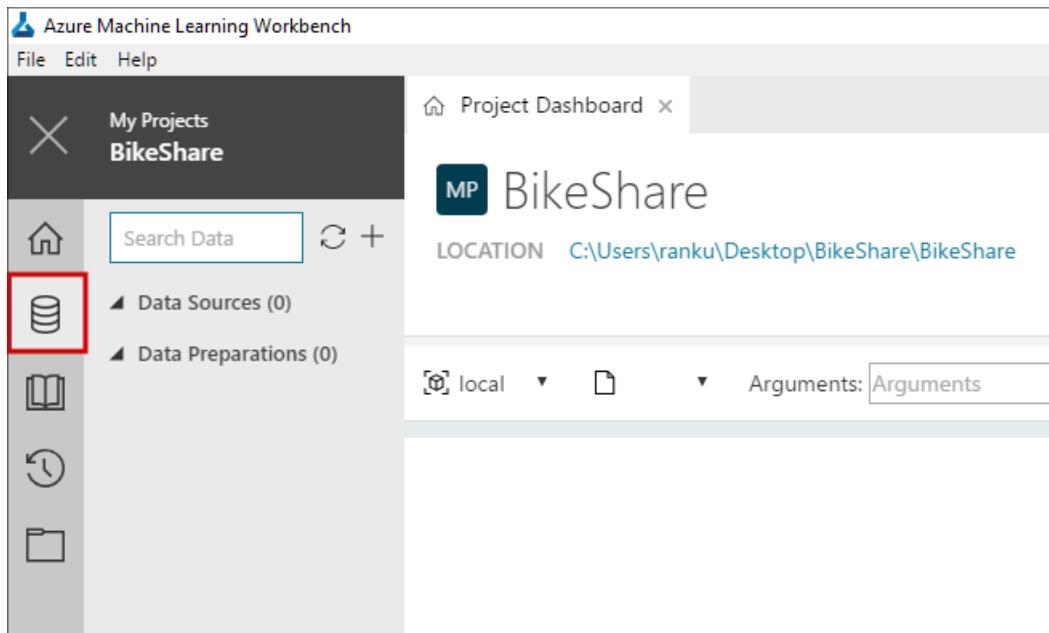
1. Start **Machine Learning Workbench** from your Start menu or launcher.

2. Create a new Machine Learning project. Select the + button on the **Projects** page, or select **File > New**.

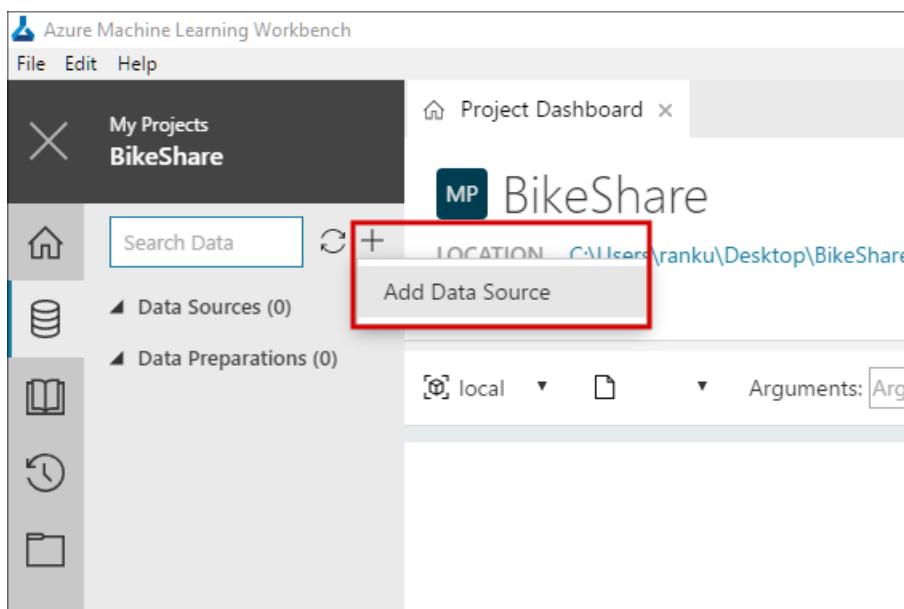
- Use the **Bike Share** template.
- Name your project **BikeShare**.

Create a new data source

1. Create a new data source. Select the **Data** button (cylinder icon) on the left toolbar to display the **Data** view.



2. Add a data source. Select the + icon, and then select **Add Data Source**.



Add weather data

1. **Data Store:** Select the data store that contains the data. Because you're using files, select **File(s)/Directory**. Select **Next** to continue.

Add Data Source

1. Data Store

Where does the data come from?

Specify the data store where the data comes from.

2. File Selection

File(s)/Directory

Database

3. File Details

4. Data Types

5. Sampling

6. Path Column

2. **File Selection:** Add the weather data. Browse and select the `BostonWeather.csv` file that you uploaded to Blob Storage earlier. Select **Next**.

Browse to find the file

Browse to the path of the file you would like to use.

Path

Azure Blob

<https://bikesharestorage.blob.core.windows.net/data-files/weather/BostonWeather.csv?st=2017-11-22>

[Browse...](#)

3. **File Details:** Verify the file schema that is detected. Machine Learning Workbench analyzes the data in the file and infers the schema to use.

Add Data Source

1. Data Store

Choose file parameters
Set parameters to interpret the file.

File Type
Delimited File (csv, tsv, txt, etc.)

Separator Comma [,] **Comment Line Character**

2. File Selection

3. File Details

4. Data Types

Skip Lines Mode
Don't skip

5. Sampling

File Encoding
utf-8

6. Path Column

Promote Headers Mode
Use Headers From First File

	abc Path	abc DATE	abc REPORTTYPE	abc HOURLYDRYBULBTEMPE	abc HOURLYRelativeHumidity	abc HOURLYWindSpeed
1	C:\Users\ranku...	1/1/2015 0:54	FM-15	22	50	10
2	C:\Users\ranku...	1/1/2015 1:00	FM-12	22	50	10
3	C:\Users\ranku...	1/1/2015 1:54	FM-15	22	50	10
4	C:\Users\ranku...	1/1/2015 2:54	FM-15	22	50	11
5	C:\Users\ranku...	1/1/2015 3:54	FM-15	24	46	13
6	C:\Users\ranku...	1/1/2015 4:00	FM-12	24	46	13
7	C:\Users\ranku...	1/1/2015 4:54	FM-15	22	52	15
8	C:\Users\ranku...	1/1/2015 5:54	FM-15	23	48	17
...

Previous **Next** **Finish**

IMPORTANT

Workbench might not detect the correct schema in some cases. Always verify that the parameters are correct for your data set. For the weather data, verify that they are set to the following values:

- **File Type:** Delimited File (csv, tsv, txt, etc.)
- **Separator:** Comma [,]
- **Comment Line Character:** No value is set.
- **Skip Lines Mode:** Don't skip
- **File Encoding:** utf-8
- **Promote Headers Mode:** Use Headers From First File

The preview of the data should display the following columns:

- **Path**
- **DATE**
- **REPORTTYPE**
- **HOURLYDRYBULBTEMPE**
- **HOURLYRelativeHumidity**
- **HOURLYWindSpeed**

To continue, select **Next**.

4. **Data Types:** Review the data types that are detected automatically. Machine Learning Workbench analyzes the data in the file and infers the data types to use.

a. For this data, change **DATA TYPE** for all the columns to **String**.

NOTE

String is used to highlight the capabilities of Workbench later in this tutorial.

Add Data Source

Set the types of your data
Set the type for the columns in your data.

Show Numeric (0) Date (0) Boolean (0) String (6)

COLUMN NAME	DATA TYPE
Path	String
DATE	String
REPORTTYPE	String
HOURLYDRYBULBTEMPF	String
HOURLYRelativeHumidity	String
HOURLYWindSpeed	String

SAMPLE OUTPUT DATA

C:\Users\ranku\Desktop\BikeData\BostonWeather.csv
C:\Users\ranku\Desktop\BikeData\BostonWeather.csv
C:\Users\ranku\Desktop\BikeData\BostonWeather.csv
1/1/2015 0:54
1/1/2015 1:00
1/1/2015 1:54
FM-15
FM-12
FM-15
22
22
22
50
50
50
10
10
10

Previous **Next** Finish

b. To continue, select **Next**.

5. **Sampling:** To create a sampling scheme, select **Edit**. Select the new **Top 10000** row that is added, and then select **Edit**. Set **Sample Strategy** to **Full File**, and then select **Apply**.

Data sampling

You can choose to bring in the entire file for completeness or a sample for better performance.

+ New		★ Set as Active	Edit	Delete
SAMPLE NAME	STRATEGY	DETAILS		
★ Top 10000	Top	Count=10000		
Sample Strategy				
<input checked="" type="button"/> Full File				
Apply Cancel				

To use the **Full File** strategy, select the **Full File** entry, and then select **Set as Active**. A star appears next to **Full File** to indicate that it's the active strategy.

Add Data Source

1. Data Store

2. File Selection

3. File Details

4. Data Types

5. Sampling

6. Path Column

Data sampling

You can choose to bring in the entire file for completeness or a sample for better performance.

+ New Set as Active Edit Delete

SAMPLE NAME	STRATEGY	DETAILS
Top 10000	Top	Count=10000
★ Full File	Full File	

To continue, select **Next**.

6. **Path Column:** Use the **Path Column** section to include the full file path as a column in the imported data. Select **Do Not Include Path Column**.

TIP

Including the path as a column is useful if you're importing a folder of many files with different file names. It's also useful if the file names contain information that you want to extract later.

Add Data Source

7. **Finish:** To finish creating the data source, select **Finish**.

A new data source tab named **BostonWeather** opens. A sample of the data is displayed in a grid view. The sample is based on the active sampling scheme specified previously.

Notice that the **Steps** pane on the right side of the screen displays the individual actions taken while creating this data source.

The screenshot shows the Azure Machine Learning Workbench interface. In the center, there is a grid view of data with columns labeled DATE, REPORTTYPE, HOURLYDRYBULBTEM, HOURLYRelativeHumid, and HOURLYWindSpeed. On the left, there's a sidebar with icons for Home, Search Data, and Data Sources. The Data Sources section lists 'BostonWeather' under 'Data Sources (1)'. On the right, a red box highlights the 'STEPS' sidebar, which contains five items: Load (BostonWeather.csv), Parse delimited, Convert Field Types, Sample, and Handle Path Column.

View data source metrics

Select **Metrics** at the upper left of the tab's grid view. This view displays the distribution and other aggregated statistics of the sampled data.

The screenshot shows the Metrics view with a red box around the 'Choose Metric' dropdown. The table displays various metrics for columns such as DATE, REPORTTYPE, HOURLYDRYBULBTEM, HOURLYRelativeHumid, and HOURLYWindSpeed. Each row includes a histogram, count, number of missing values, column data type, numerical column, number of NaNs, most common value, and count of most common value.

Column	Profile	Count	Number of missing val	Column data type	Numerical column	Number of NaNs	Most common	Count of most common	Number of unique values
DATE	11/15/2015 9:54 3/15/2016 13:00 1/31/2016 4:54	30076	0	object	false	30076	11/15/2015 9:54	1	30076
REPORTTYPE	FM-15 FM-12 FM-16	30076	0	object	false	30076	FM-15	18943	5
HOURLYDRYBULBTEM	NA 39 46	30076	0	object	false	796	NA	791	109
HOURLYRelativeHumid	100 93 NA	30076	0	object	false	794	100	1743	89
HOURLYWindSpeed	9 8 10	30076	0	object	false	800	9	2613	32

NOTE

To configure the visibility of the statistics, use the **Choose Metric** drop-down list. Select and clear metrics there to change the grid view.

To return to the **Data** view, select **Data** in the upper left of the page.

Add a data source to the data preparation package

1. Select **Prepare** to begin preparing the data.
2. When prompted, enter a name for the data preparation package, such as **BikeShare Data Prep**.
3. Select **OK** to continue.



4. A new package named **BikeShare Data Prep** appears under the **Data Preparation** section of the **Data** tab.

To display the package, select this entry.

5. Select the **>>** button to expand **Dataflows** and display the dataflows contained in the package. In this example, **BostonWeather** is the only dataflow.

IMPORTANT

A package can contain multiple dataflows.

	DATE	REPORTTYPE
1	1/1/2015 0:54	FM-15
2	1/1/2015 1:00	FM-12
3	1/1/2015 1:54	FM-15
4	1/1/2015 2:54	FM-15
5	1/1/2015 3:54	FM-15
6	1/1/2015 4:00	FM-12
7	1/1/2015 4:54	FM-15

Filter data by value

1. To filter data, right-click a cell with a certain value, and select **Filter**. Then select the type of filter.
2. For this tutorial, select a cell that contains the value **FM-15**. Then set the filter to **equals**. Now the data is filtered to only return rows where the **REPORTTYPE** is **FM-15**.

DATAFLOWS << BostonWeather Metrics

BostonWeather	abc DATE	abc REPORTTP...	abc HOURLYD...	abc HOURLYRe...	abc HOURLYW...
1	1/1/2015 0:54	FM-15	22	50	10
2	1/1/2015 1:00	FM-12	22	50	10
3	1/1/2015 1:54	FM-15	22	50	10
4	1/1/2015 2:54	FM-15	22	50	11
5	1/1/2015 3:54	FM-15	24	50	13
6	1/1/2015 4:00	FM-12	24	50	13
7	1/1/2015 4:54	FM-15	22	50	15
8	1/1/2015 5:54	FM-15	23	50	17
9	1/1/2015 6:54	FM-15	23	50	14
10	1/1/2015 7:00	FM-12	23	50	14
11	1/1/2015 7:54	FM-15	22	52	13
12	1/1/2015 8:54	FM-15	25	44	16
13	1/1/2015 9:54	FM-15	28	39	14
14	1/1/2015 10:00	FM-12	28	39	14

NOTE

FM-15 is a type of Meteorological Terminal Aviation Routine (METAR) weather report. The FM-15 reports are empirically observed to be the most complete, with little missing data.

Remove a column

You no longer need the **REPORTTYPE** column. Right-click the column header, and select **Remove Column**.

Valid: 18943	Missing: 0	Error: 0
REPORTTPYE		
	abc REPORTTP...	abc HOURLYD...
	FM-15	Derive Column by Example
	FM-15	Split Column by Example
	FM-15	Expand JSON
	FM-15	Duplicate Column
	FM-15	Text Clustering
	FM-15	Replace Values
	FM-15	Replace NA Values
	FM-15	Trim String
	FM-15	Handle Missing Values
	FM-15	Handle Error Values
	FM-15	Rename Column
	FM-15	Remove Column
	FM-15	Keep Column
	FM-15	Convert Field Type to Numeric
	FM-15	Convert Field Type to Date
	FM-15	Convert Field Type to Boolean

Change datatypes and remove errors

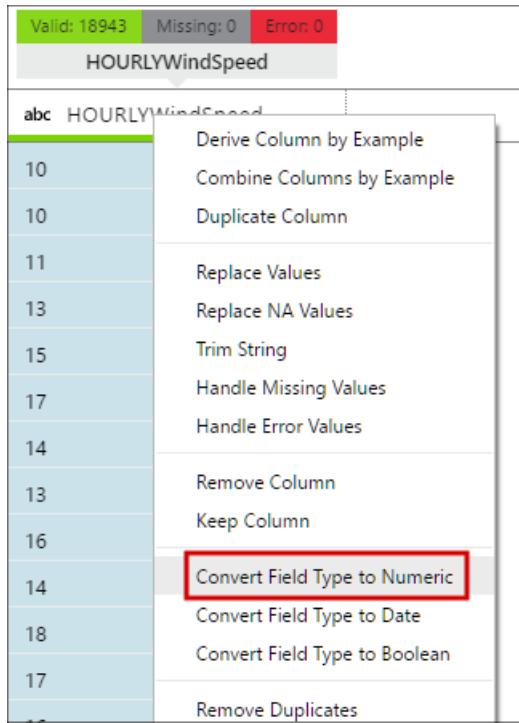
1. Select Ctrl (Command ⌘ on Mac) while you select column headers to select multiple columns at the same time. Use this technique to select the following column headers:

- **HOURLYDRYBULBTEMPP**

- **HOURLYRelativeHumidity**

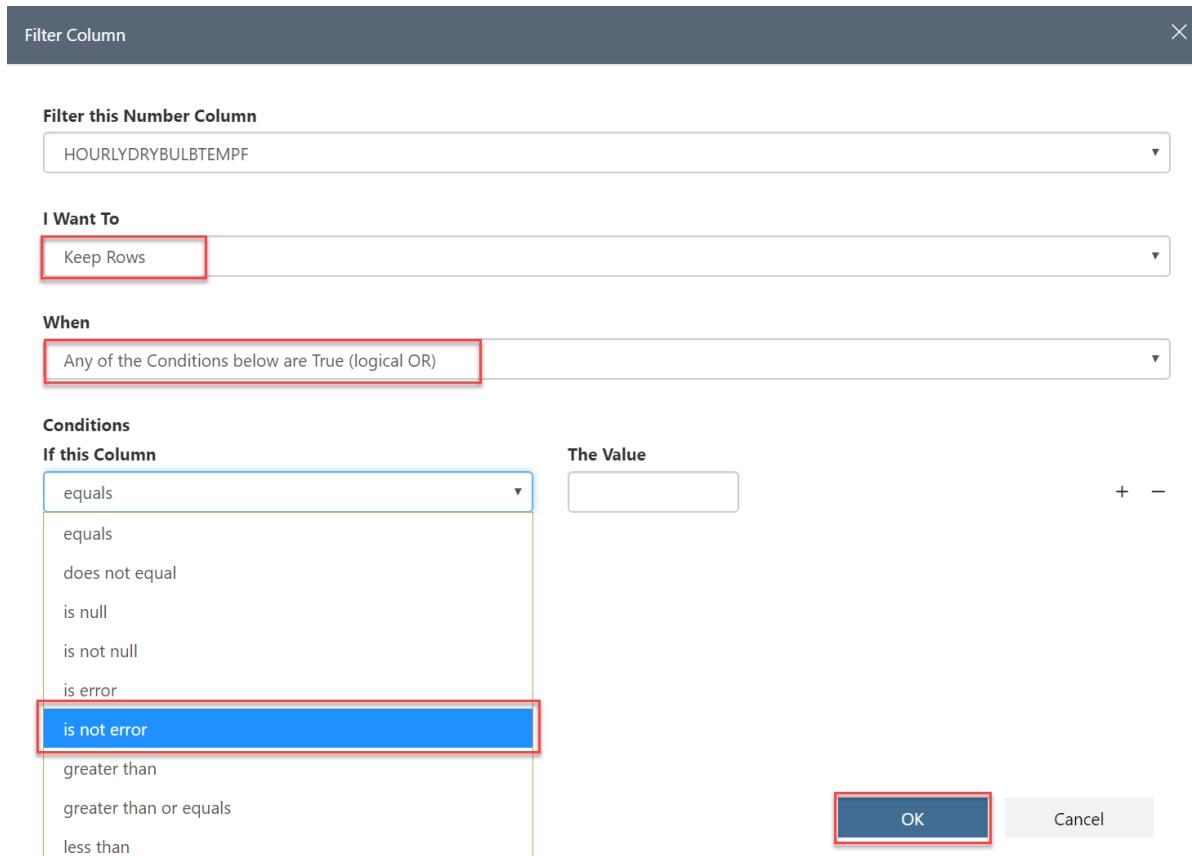
- **HOURLYWindSpeed**

2. Right-click one of the selected column headers, and select **Convert Field Type to Numeric**. This option converts the data type for the columns to numeric.



3. Filter out the error values. Some columns have data type conversion problems. This problem is indicated by the red color in the **Data Quality Bar** for the column.

To remove the rows that have errors, right-click the **HOURLYDRYBULBTEMPF** column heading. Select **Filter Column**. Use the default **I Want To** as **Keep Rows**. Change the **Conditions** drop-down list to select **is not error**. Select **OK** to apply the filter.



- To eliminate the remaining error rows in the other columns, repeat this filter process for the **HOURLYRelativeHumidity** and **HOURLYWindSpeed** columns.

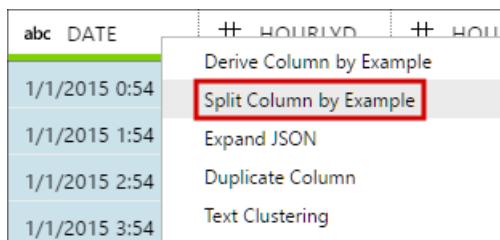
Use by example transformations

To use the data in a prediction for two-hour time blocks, you must compute the average weather conditions for two-hour periods. Use the following actions:

- Split the **DATE** column into separate **Date** and **Time** columns. See the following section for the detailed steps.
- Derive an **Hour_Range** column from the **Time** column. See the following section for the detailed steps.
- Derive a **Date_Hour_Range** column from the **DATE** and **Hour_Range** columns. See the following section for the detailed steps.

Split column by example

- Split the **DATE** column into separate **Date** and **Time** columns. Right-click the **DATE** column header, and select **Split Column by Example**.



- Machine Learning Workbench automatically identifies a meaningful delimiter and creates two columns by splitting the data into date and time values.
- Select **OK** to accept the split operation results.

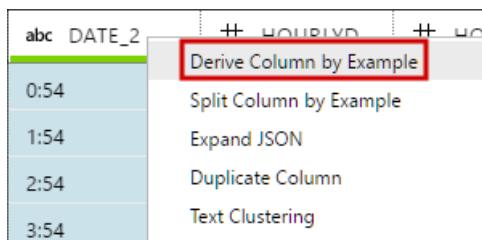
A screenshot of the Machine Learning Workbench interface showing the 'BostonWeather' dataset. The table has the following structure:

	abc DATE	abc DATE_1	abc DATE_2	# HOURLYD...	# HOURLYRe...	# HOURLYW...
1	1/1/2015 0:54	1/1/2015	0:54	22	50	10
2	1/1/2015 1:54	1/1/2015	1:54	22	50	10
3	1/1/2015 2:54	1/1/2015	2:54	22	50	11
4	1/1/2015 3:54	1/1/2015	3:54	24	46	13

The 'OK' button at the top right of the dialog is highlighted with a red box.

Derive column by example

- Derive a two-hour range, right-click the **DATE_2** column header, and select **Derive Column by Example**.



A new empty column is added with null values.

- Select in the first empty cell in the new column. To provide an example of the time range desired, type

12AM-2AM in the new column, and then select Enter.

BostonWeather Metrics							
DERIVE COLUMN BY EXAMPLE: You have selected 1 source column and provided 0 examples. No suggestions Advanced mode							
	<input type="checkbox"/> abc DATE	<input type="checkbox"/> abc DATE_1	<input checked="" type="checkbox"/> abc DATE_2	<input type="checkbox"/> abc Column	<input type="checkbox"/> # HOURLYD...	<input type="checkbox"/> # HOURLYRe...	<input type="checkbox"/> # HOURLYW...
1	1/1/2015 0:54	1/1/2015	0:54	12AM-2AM	22	50	10
2	1/1/2015 1:54	1/1/2015	1:54	null	22	50	10
3	1/1/2015 2:54	1/1/2015	2:54	null	22	50	11

NOTE

Machine Learning Workbench synthesizes a program based on the examples provided by you and applies the same program on remaining rows. All other rows are automatically populated based on the example you provided.

Workbench also analyzes your data and tries to identify edge cases.

IMPORTANT

Identification of edge cases might not work on Mac in the current version of Workbench. Skip the following step 3 and step 4 on Mac. Instead, select **OK** after all the rows are populated with the derived values.

3. The text **Analyzing Data** above the grid indicates that Workbench is trying to detect edge cases. When finished, the status changes to **Review next suggested row** or **No suggestions**. In this example, **Review next suggested row** is returned.
4. To review the suggested changes, select **Review next suggested row**. The cell that you should review and correct, if needed, is highlighted on the display.

BostonWeather Metrics							
DERIVE COLUMN BY EXAMPLE: You have selected 1 source column and provided 1 example. Review next suggested row Advanced mode							
	<input type="checkbox"/> abc DATE	<input type="checkbox"/> abc DATE_1	<input checked="" type="checkbox"/> abc DATE_2	<input type="checkbox"/> abc Column	<input type="checkbox"/> # HOURLYD...	<input type="checkbox"/> # HOURLYRe..	<input type="checkbox"/> # HOURLYW...
1	1/1/2015 0:54	1/1/2015	0:54	12AM-2AM	22	50	10
2	1/1/2015 1:54	1/1/2015	1:54	12AM-2AM	22	50	10
3	1/1/2015 2:54	1/1/2015	2:54	2AM-4AM	22	50	11
4	1/1/2015 3:54	1/1/2015	3:54	2AM-4AM	24	46	13

Select **OK** to accept the transformation.

5. You are returned to the grid view of data for **BostonWeather**. The grid now contains the three columns added previously.

BostonWeather Metrics				
	abc DATE	abc DATE_1	abc DATE_2	abc Column
1	1/1/2015 0:54	1/1/2015	0:54	12AM-2AM
2	1/1/2015 1:54	1/1/2015	1:54	12AM-2AM
3	1/1/2015 2:54	1/1/2015	2:54	2AM-4AM
4	1/1/2015 3:54	1/1/2015	3:54	2AM-4AM

TIP

All the changes you made are preserved in the **Steps** pane. Go to the step that you created in the **Steps** pane, select the down arrow, and select **Edit**. The advanced window for **Derive Column by Example** is displayed. All your examples are preserved here. You also can add examples manually by double-clicking on a row in the following grid. Select **Cancel** to return to the main grid without applying changes. You also can access this view by selecting **Advanced mode** while you perform a **Derive Column by Example** transform.

6. To rename the column, double-click the column header, and type **Hour Range**. Select Enter to save the change.

BostonWeather Metrics				
	abc DATE	abc DATE_1	abc DATE_2	abc Hour Range
1	1/1/2015 0:54	1/1/2015	0:54	12AM-2AM
2	1/1/2015 1:54	1/1/2015	1:54	12AM-2AM
3	1/1/2015 2:54	1/1/2015	2:54	2AM-4AM
4	1/1/2015 3:54	1/1/2015	3:54	2AM-4AM

7. To derive the date and hour range, multi-select the **Date_1** and **Hour Range** columns, right-click, and then select **Derive Column by Example**.

BostonWeather Metrics					Valid: 18935	Missing: 0	Error: 0
					Hour Range		
	abc DATE	abc DATE_1	abc DATE_2	abc Hour Range	++ HOURWD	++ HU	
1	1/1/2015 0:54	1/1/2015	0:54	12AM-2AM			Derive Column by Example
2	1/1/2015 1:54	1/1/2015	1:54	12AM-2AM			Combine Columns by Example

Type **Jan 01, 2015 12AM-2AM** as the example against the first row, and select Enter.

Workbench determines the transformation based on the example you provide. In this example, the result is that the date format is changed and concatenated with the two-hour window.

<input checked="" type="checkbox"/> abc Hour Range	<input type="checkbox"/> abc Column
12AM-2AM	Jan 01, 2015 12AM-2AM
12AM-2AM	null
2AM-4AM	null
2AM-4AM	null

IMPORTANT

On a Mac, take the following step instead of step 8:

- Go to the first cell that contains **Feb 01, 2015 12AM-2AM**. It should be row 15. Correct the value to **Jan 02, 2015 12AM-2AM**, and select Enter.

8. Wait for the status to change from **Analyzing Data** to **Review next suggested row**. This change might take several seconds. Select the status link to go to the suggested row.

8PM-10PM	Dec 01, 2015 8PM-10PM
10PM-12AM	Dec 01, 2015 10PM-12AM
10PM-12AM	Dec 01, 2015 10PM-12AM
12AM-2AM	null
12AM-2AM	null
2AM-4AM	null

The row has a null value because the source date value can be for either dd/mm/yyyy or mm/dd/yyyy. Type the correct value of **Jan 13, 2015 2AM-4AM**, and select Enter. Workbench uses the two examples to improve the derivation for the remaining rows.

12AM-2AM	Jan 13, 2015 12AM-2AM
2AM-4AM	Jan 13, 2015 2AM-4AM
2AM-4AM	Jan 13, 2015 2AM-4AM
4AM-6AM	Jan 13, 2015 4AM-6AM

9. Select **OK** to accept the transform.

BostonWeather Metrics					
	abc DATE	abc DATE_1	abc DATE_2	abc Hour Range	abc Column
1	1/1/2015 0:54	1/1/2015	0:54	12AM-2AM	Jan 01, 2015 12AM-2AM
2	1/1/2015 1:54	1/1/2015	1:54	12AM-2AM	Jan 01, 2015 12AM-2AM
3	1/1/2015 2:54	1/1/2015	2:54	2AM-4AM	Jan 01, 2015 2AM-4AM
4	1/1/2015 3:54	1/1/2015	3:54	2AM-4AM	Jan 01, 2015 2AM-4AM
5	1/1/2015 4:54	1/1/2015	4:54	4AM-6AM	Jan 01, 2015 4AM-6AM

TIP

To use the **Advanced mode of Derive Column by Example** for this step, select the down arrow in the **Steps** pane. In the data grid, there are check boxes next to the **DATE_1** and **Hour Range** columns. Clear the check box next to the **Hour Range** column to see how the output changes. In the absence of the **Hour Range** column as input, **12AM-2AM** is treated as a constant and is appended to the derived values. Select **Cancel** to return to the main grid without applying your changes.

BostonWeather

DERIVE COLUMN BY EXAMPLE: You have selected 2 source columns and provided 2 examples. [Basic mode](#)

Output Column Name
Column

abc DATE_1	abc Hour_Range
1/1/2015	12AM-2AM
1/13/2015	2AM-4AM

There are no suggested examples to show

◀ ▶

Hint: Double-click on a row below to add it...

OK

Cancel

<input checked="" type="checkbox"/> abc DATE_1	<input type="checkbox"/> abc DATE_2	<input checked="" type="checkbox"/> abc Hour_Ran...	<input type="checkbox"/> abc Column
1/1/2015	0:54	12AM-2AM	Jan 01, 2015 12AM-2AM
1/1/2015	1:54	12AM-2AM	Jan 01, 2015 12AM-2AM

10. To rename the column, double-click the header. Change the name to **Date Hour Range**, and then select Enter.
11. Multi-select the **DATE**, **DATE_1**, **DATE_2**, and **Hour Range** columns. Right-click, and then select **Remove column**.

Summarize data (mean)

The next step is to summarize the weather conditions by taking the mean of the values, grouped by hour range.

1. Select the **Date Hour Range** column, and then on the **Transforms** menu, select **Summarize**.

Transforms Inspectors View Help

BostonWeather BikeShare Data Prep

BostonWeather Metrics

	abc Date Hour Range	# HOURLYD...
1	Jan 01, 2015 12AM-2AM	22
2	Jan 01, 2015 12AM-2AM	22
3	Jan 01, 2015 2AM-4AM	22
4	Jan 01, 2015 2AM-4AM	24
5	Jan 01, 2015 4AM-6AM	22
6	Jan 01, 2015 4AM-6AM	23
7	Jan 01, 2015 6AM-8AM	23
8	Jan 01, 2015 6AM-8AM	22
9	Jan 01, 2015 8AM-10AM	25
10	Jan 01, 2015 8AM-10AM	28
11	Jan 01, 2015 10AM-12PM	30
12	Jan 01, 2015 10AM-12PM	31
13	Jan 01, 2015 12PM-2PM	32
14	Jan 01, 2015 12PM-2PM	33
15	Jan 01, 2015 2PM-4PM	33
16	Jan 01, 2015 2PM-4PM	31
17	Jan 01, 2015 4PM-6PM	31
18	Jan 01, 2015 4PM-6PM	30
19	Jan 01, 2015 6PM-8PM	31
20	Jan 01, 2015 6PM-8PM	30

Derived Column by Example
Split Column by Example
Expand JSON
Combine Columns by Example
Duplicate Column
Text Clustering
Replace Values
Replace NA Values
Trim String
Handle Missing Values
Handle Error Values
Adjust Precision
Rename Column
Remove Column
Keep Column
Convert Field Type to Numeric
Convert Field Type to Date
Convert Field Type to Boolean
Convert Field Type to String
Convert Unix Timestamp to DateTime
Filter Column
Use First Row as Headers
Join
Append Rows
Append Columns
Summarize
Remove Duplicates

- To summarize the data, drag columns from the grid at the bottom of the page to the left and right panes at the top. The left pane contains the text **Drag columns here to group data**. The right pane contains the text **Drag columns here to summarize data**.
 - Drag the **Date Hour Range** column from the grid at the bottom to the left pane. Drag **HOURLYDRYBULBTEMPF**, **HOURLYRelativeHumidity**, and **HOURLYWindSpeed** to the right pane.
 - In the right pane, select **Mean** as the **Aggregate** measure for each column. Select **OK** to finish the summarization.

The screenshot shows the 'Metrics' pane of the Power BI Data Flow interface. On the left, there's a 'Group By' section containing a field named 'Date Hour Ran...'. To the right, there are three 'Aggregate' sections, each with a dropdown set to 'Mean'. These sections correspond to columns: 'HOURLYDRYBULBTEMPF_Mean', 'HOURLYRelativeHumidity_Mean', and 'HOURLYWindSpeed_Mean'. Below these, there's a 'New Column Name' section where the names are displayed. At the bottom right of the pane is a large blue 'OK' button.

Transform dataflow by using script

Changing the data in the numeric columns to a range of 0 to 1 allows some models to converge quickly. Currently, there is no built-in transformation to generically do this transformation. Use a Python script to perform this operation.

1. On the **Transform** menu, select **Transform Dataflow (Script)**.
2. Enter the following code in the text box that appears. If you used the column names, the code should work without modification. You are writing a simple min-max normalization logic in Python.

WARNING

The script expects the column names used previously in this tutorial. If you have different column names, you must change the names in the script.

```
maxVal = max(df["HOURLYDRYBULBTEMPF_Mean"])
minVal = min(df["HOURLYDRYBULBTEMPF_Mean"])
df["HOURLYDRYBULBTEMPF_Mean"] = (df["HOURLYDRYBULBTEMPF_Mean"]-minVal)/(maxVal-minVal)
df.rename(columns={"HOURLYDRYBULBTEMPF_Mean":"N_DryBulbTemp"},inplace=True)

maxVal = max(df["HOURLYRelativeHumidity_Mean"])
minVal = min(df["HOURLYRelativeHumidity_Mean"])
df["HOURLYRelativeHumidity_Mean"] = (df["HOURLYRelativeHumidity_Mean"]-minVal)/(maxVal-minVal)
df.rename(columns={"HOURLYRelativeHumidity_Mean":"N_RelativeHumidity"},inplace=True)

maxVal = max(df["HOURLYWindSpeed_Mean"])
minVal = min(df["HOURLYWindSpeed_Mean"])
df["HOURLYWindSpeed_Mean"] = (df["HOURLYWindSpeed_Mean"]-minVal)/(maxVal-minVal)
df.rename(columns={"HOURLYWindSpeed_Mean":"N_WindSpeed"},inplace=True)

df
```

TIP

The Python script must return `df` at the end. This value is used to populate the grid.

Transform Dataflow (Script)

Code to Transform Dataflow

```
1 maxVal = max(df["HOURLYDRYBULBTEMPF_Mean"])
2 minVal = min(df["HOURLYDRYBULBTEMPF_Mean"])
3 df["HOURLYDRYBULBTEMPF_Mean"] = (df["HOURLYDRYBULBTEMPF_Mean"]-minVal)/(maxVal-minVal)
4 df.rename(columns={"HOURLYDRYBULBTEMPF_Mean":"N_DryBulbTemp"},inplace=True)
5
6 maxVal = max(df["HOURLYRelativeHumidity_Mean"])
7 minVal = min(df["HOURLYRelativeHumidity_Mean"])
8 df["HOURLYRelativeHumidity_Mean"] = (df["HOURLYRelativeHumidity_Mean"]-minVal)/(maxVal-minVal)
9 df.rename(columns={"HOURLYRelativeHumidity_Mean":"N_RelativeHumidity"},inplace=True)
10
11 maxVal = max(df["HOURLYWindSpeed_Mean"])
12 minVal = min(df["HOURLYWindSpeed_Mean"])
13 df["HOURLYWindSpeed_Mean"] = (df["HOURLYWindSpeed_Mean"]-minVal)/(maxVal-minVal)
14 df.rename(columns={"HOURLYWindSpeed_Mean":"N_WindSpeed"},inplace=True)
```

Code Block Type

Expression

Hint

Please provide Python (3.5) code that transforms your data.
A Pandas DataFrame called 'df' has been made available to your code. Your code should either modify 'df' or reassign a new DataFrame to 'df' before it completes.
The following Python imports are provided: math, numbers, datetime, re, pandas (aliased as pd), numpy (aliased as np), scipy (aliased as sp).
Module signature: def transform(df): return transformed Pandas DataFrame.

OK Cancel

3. Select **OK** to use the script. The numeric columns in the grid now contain values in the range of 0 to 1.

BostonWeather				
	abc Date Hour...	# N_DryBul...	# N_Relative...	# N_WindSp...
1	Jan 01, 2015 12..	0.29383886255...	0.42528735632...	0.28169014084...
2	Jan 01, 2015 2...	0.30331753554...	0.40229885057...	0.33802816901...
3	Jan 01, 2015 4...	0.29857819905...	0.42528735632...	0.45070422535...
4	Jan 01, 2015 6...	0.29857819905...	0.43678160919...	0.38028169014...
5	Jan 01, 2015 8...	0.33649289099...	0.32758620689...	0.42253521126...

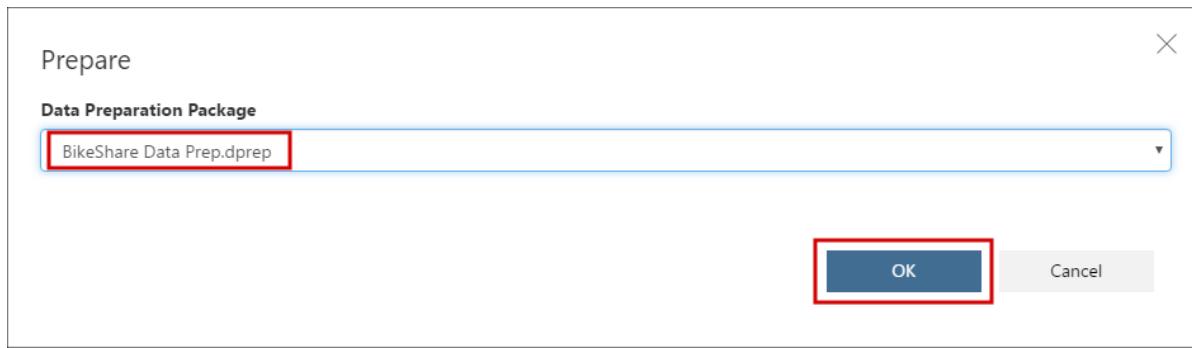
You have finished preparing the weather data. Next, prepare the trip data.

Load trip data

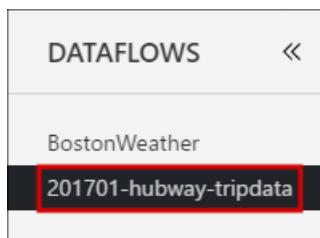
1. To import the `201701-hubway-tripdata.csv` file, use the steps in the [Create a new data source](#) section. Use the following options during the import process:
 - **File Selection:** Select **Azure Blob** when you browse to select the file.
 - **Sampling Scheme:** Select **Full File** sampling scheme, and make the sample active.
 - **Data Type:** Accept the defaults.

2. After you import the data, select **Prepare** to begin preparing the data. Select the existing **BikeShare Data Prep.dprep** package, and then select **OK**.

This process adds a **Dataflow** to the existing **Data Preparation** file rather than creating a new one.



3. After the grid has loaded, expand **DATAFLOWS**. There are now two dataflows: **BostonWeather** and **201701-hubway-tripdata**. Select the **201701-hubway-tripdata** entry.



Use the map inspector

For data preparation, useful visualizations called inspectors are available for string, numeric, and geographical data. They help you to understand the data better and identify outliers. Follow these steps to use the map inspector.

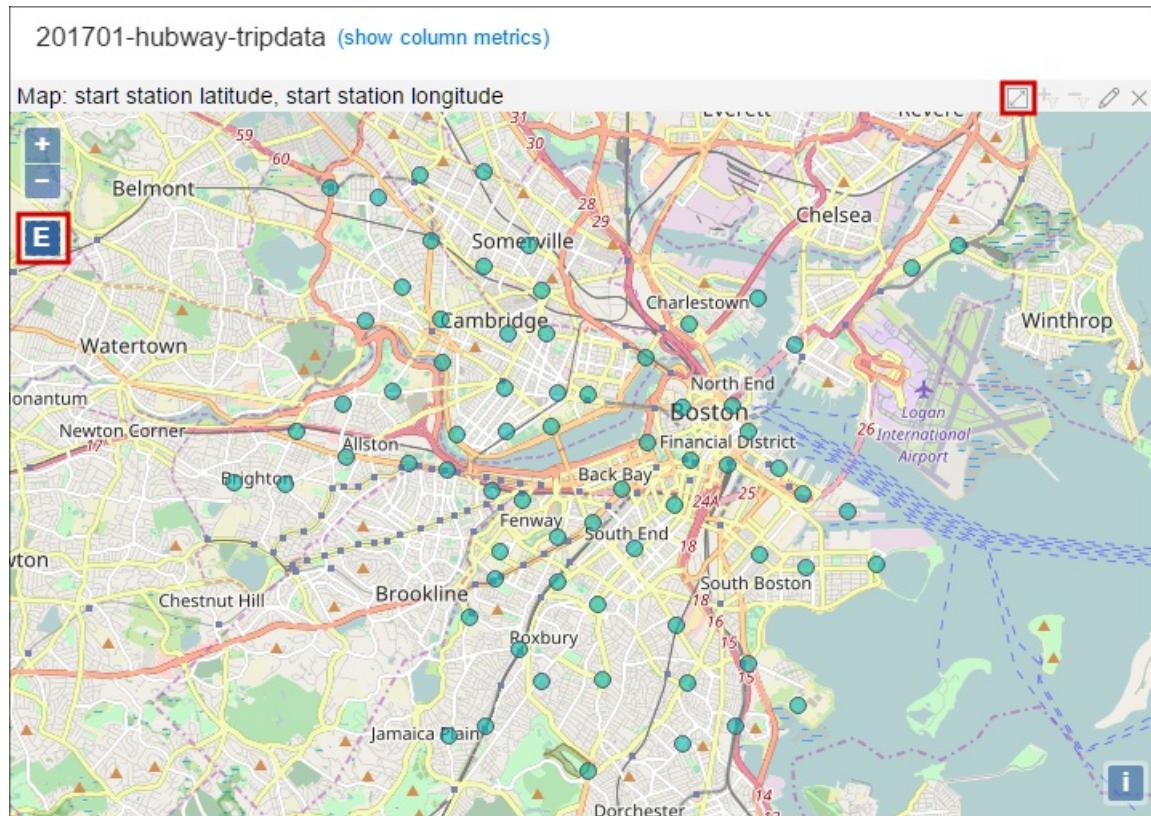
1. Multi-select the **start station latitude** and **start station longitude** columns. Right-click one of the columns, and then select **Map**.

TIP

To enable multi-select, hold down the Ctrl key (Command ⌘ on Mac), and select the header for each column.

abc start station id	# start station id	# start station name	# end station id	# end station name
MIT at Mass Av...	42.3581	-71.		Derive Column by Example
Boston Public L...	42.349673	-71.		Combine Columns by Example
Boston Public L...	42.349673	-71.		Duplicate Column
Christian Scien...	42.343864	-71.		Replace NA Values
B.U. Central - 7...	42.350406	-71.		Handle Missing Values
Cross St. at Ha...	42.362811	-71.		Handle Error Values
MIT at Mass Av...	42.3581	-71.		Remove Column
Ames St at Mai...	42.3625	-7		Keep Column
The Esplanade ...	42.355596	-7		Convert Field Type to String
Agganis Arena...	42.351246	-71.		Convert Unix Timestamp to DateTime
Inman Square ...	42.374035	-71.		Remove Duplicates
Harvard Law Sc...	42.379011	-71.		Sort
Green St T	42.310579	-71.		Add Column (Script)
Roxbury Crossi...	42.331184	-71.		Column Statistics
MIT Stata Cent...	42.3619622	-71.0		Histogram
South Station -...	42.352175	-71.		Value Counts
Washington St...	42.3384927928...	-71.0740		Box Plot
MIT Pacific Sta...	42.3505732010	-71.1012		Scatter Plot
				Map

2. To maximize the map visualization, select the **Maximize** icon. To fit the map to the window, select the **E** icon on the upper-left side of the visualization.



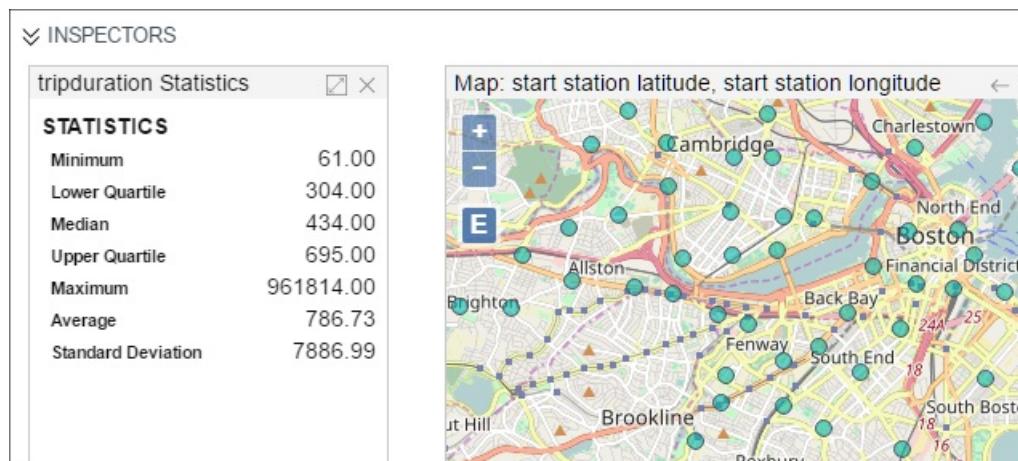
3. Select the **Minimize** button to return to the grid view.

Use the column statistics inspector

To use the column statistics inspector, right-click the **tripduration** column, and select **Column Statistics**.

#	tripduration	the starttime	the starttime
1	350	Derive Column by Example	
2	89	Split Column by Example	
3	1672	Duplicate Column	
4	747	Replace NA Values	
5	621	Handle Missing Values	
6	664	Handle Error Values	
7	260	Adjust Precision	
8	403	Rename Column	
9	642	Remove Column	
10	953	Keep Column	
11	231	Convert Field Type to String	
12	1011	Convert Unix Timestamp to DateTime	
13	1636	Filter Column	
14	196	Remove Duplicates	
15	259	Sort	
16	194	Add Column (Script)	
17	1372	Column Statistics	
18	218	Histogram	
		Value Counts	

This process adds a new visualization titled **tripduration Statistics** in the **INSPECTORS** pane.

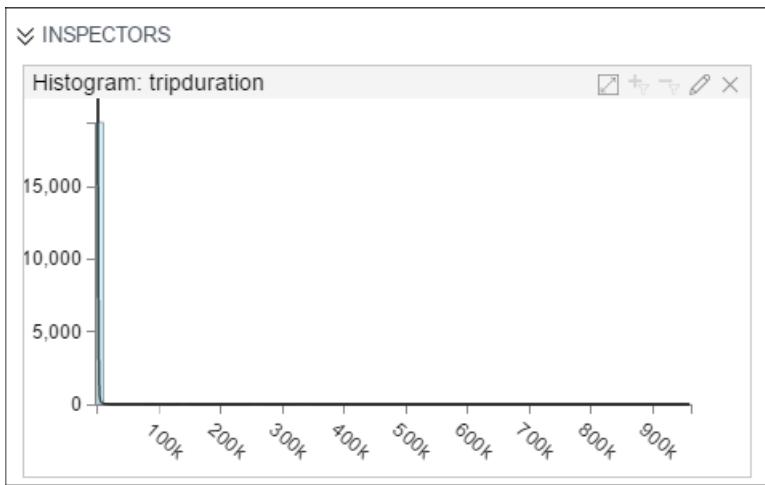


IMPORTANT

The maximum value of the trip duration is 961,814 minutes, which is about two years. It seems there are some outliers in the dataset.

Use the histogram inspector

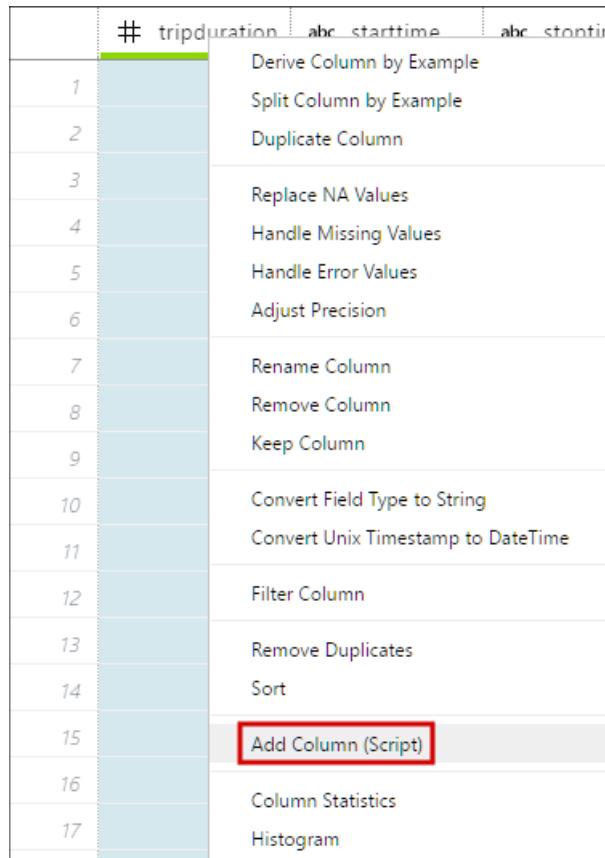
To attempt to identify outliers, right-click the **tripduration** column, and select **Histogram**.



The histogram isn't helpful because the outliers skew the graph.

Add a column by using script

1. Right-click the **tripduration** column, and select **Add Column (Script)**.



2. In the **Add Column (Script)** dialog box, use the following values:

- **New Column Name:** logtripduration
- **Insert this New Column After:** tripduration
- **New Column Code:** `math.log(row.tripduration)`
- **Code Block Type:** Expression

Edit

New Column Name
logtripduration

Insert this New Column After
tripduration

New Column Code
`math.log(row.tripduration)`

Code Block Type
Expression

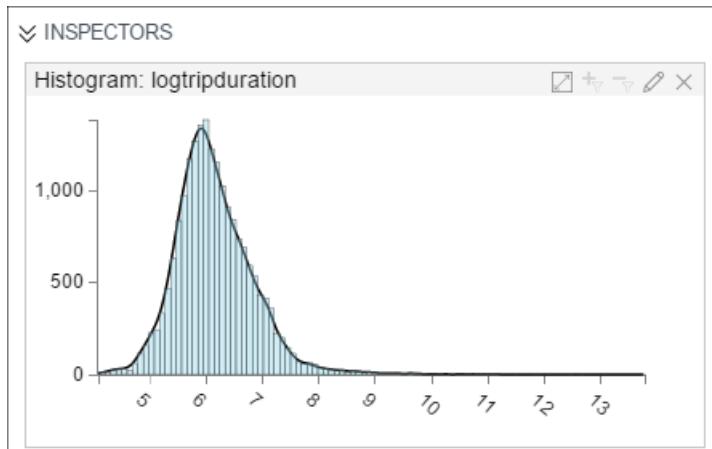
Hint
Please provide a Python (3.5) expression or module that will compute the value of the new column. The current row is referenced using 'row' and the

Examples:
`row.ColumnA + row.ColumnB` is the same as `row["ColumnA"] + row["ColumnB"]`
`1 if row.ColumnA < 4 else 2`
`datetime.datetime.now()`
`float(row.ColumnA) / float(row.ColumnB - 1)`
`'Bad' if pd.isnull(row.ColumnA) else 'Good'`

The following Python imports are provided: math, numbers, datetime, re, pandas (aliased as pd), numpy (aliased as np), scipy (aliased as sp).
Module signature: def newvalue(row): return new column value.

3. Select **OK** to add the **logtripduration** column.

4. Right-click the column, and select **Histogram**.



Visually, this histogram seems like a normal distribution with an abnormal tail.

Use an advanced filter

Using a filter on the data updates the inspectors with the new distribution.

1. Right-click the **logtripduration** column, and select **Filter Column**.

2. In the **Edit** dialog box, use the following values:

- **Filter this Number Column:** logtripduration
- **I Want To:** Keep Rows
- **When:** Any of the Conditions below are True (logical OR)
- **If this Column:** less than

- **The Value:** 9

Edit

Filter this Number Column
logtripduration

I Want To
Keep Rows

When
Any of the Conditions below are True (logical OR)

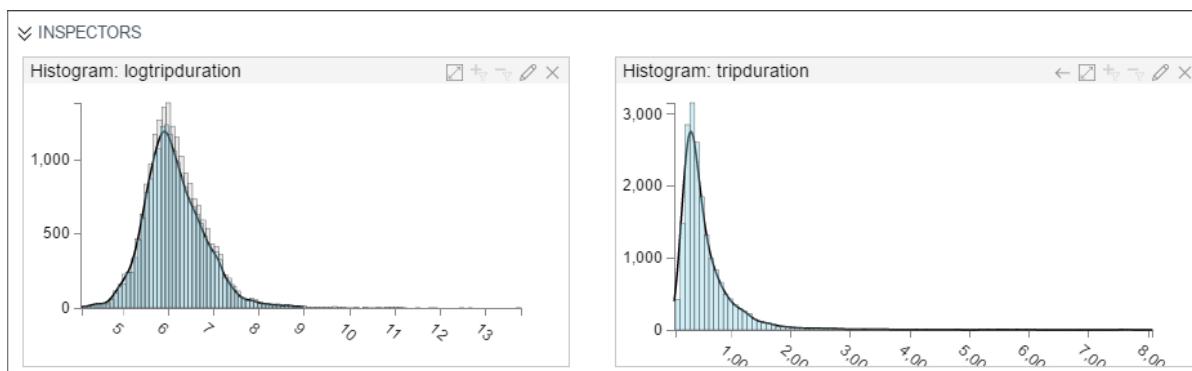
Conditions

If this Column	The Value
less than	9

+
 Create Dataflow Containing the Filtered Out Rows?

Hint
You can add Conditions by clicking on the + button.

3. Select **OK** to apply the filter.

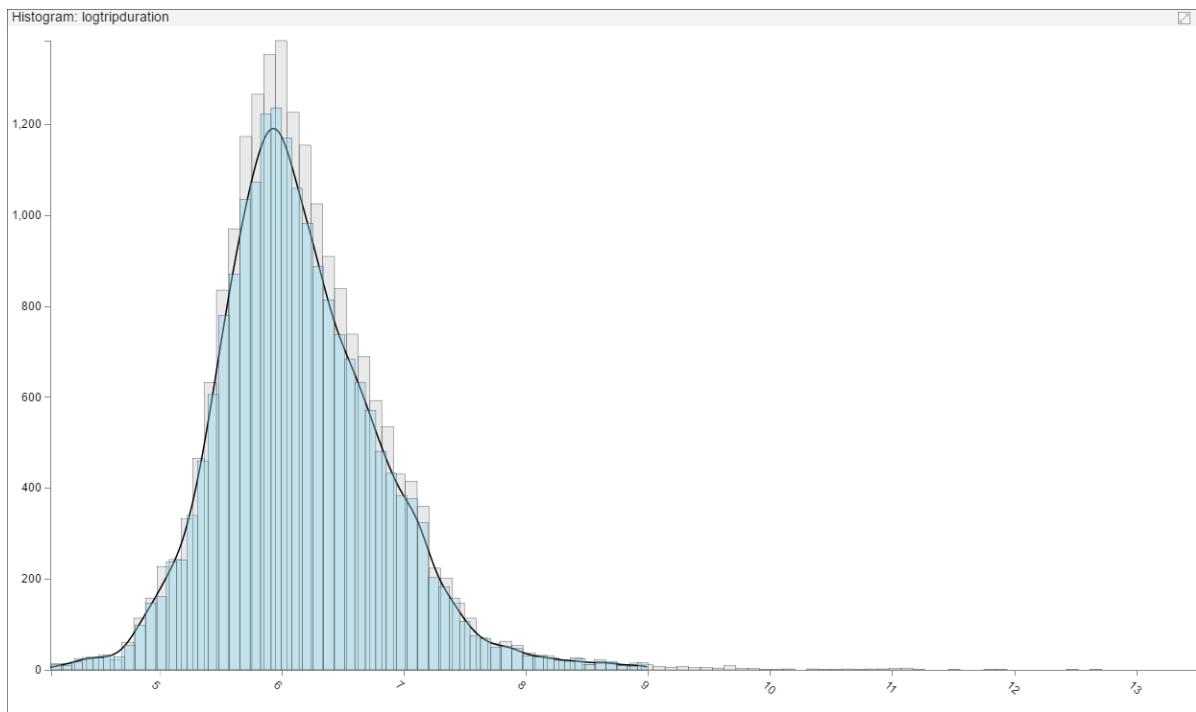


The halo effect

1. Maximize the **logtripduration** histogram. A blue histogram is overlaid on a gray histogram. This display is called the **Halo Effect**:

- The gray histogram represents the distribution before the operation (in this case, the filtering operation).
- The blue histogram represents the histogram after the operation.

The halo effect helps with visualizing the effect of an operation on the data.



NOTE

The blue histogram appears shorter compared to the previous one. This difference is due to automatic re-bucketing of data in the new range.

2. To remove the halo, select **Edit** and clear **Show halo**.

Histogram

Column

Minimum Number of Buckets (applies even when default bucketing is checked)

Default Number of Buckets (Scott's Rule)

Show halo

Kernel Density Plot Overlay (Gaussian Kernel)

3. Select **OK** to disable the halo effect. Then minimize the histogram.

Remove columns

In the trip data, each row represents a bike pickup event. For this tutorial, you need only the **starttime** and **start station id** columns. To remove the other columns, multi-select these two columns, right-click the column header, and then select **Keep Column**. Other columns are removed.

#	tripduration	starttime	stoptime	#	start statio...	abc_start statio...	# start statio...
1	350	2017-01-01 00...	2017-01-01 00...			Derive Column by Example	1
2	891	2017-01-01 00...	2017-01-01 00...			Combine Columns by Example	
3	1672	2017-01-01 00...	2017-01-01 00...			Duplicate Column	3
4	747	2017-01-01 00...	2017-01-01 00...			Replace NA Values	3
5	621	2017-01-01 00...	2017-01-01 00...			Handle Missing Values	4
6	664	2017-01-01 00...	2017-01-01 00...			Handle Error Values	6
7	260	2017-01-01 00...	2017-01-01 00...			Remove Column	1
8	403	2017-01-01 00...	2017-01-01 00...			Keep Column	1
9	642	2017-01-01 00...	2017-01-01 00...			Convert Field Type to String	5
						Convert Unix Timestamp to DateTime	6

Summarize data (count)

To summarize bike demand for a two-hour period, use derived columns.

1. Right-click the **starttime** column, and select **Derive Column by Example**.

#	starttime	Derive Column by Example
1	2017-01-01 00:06:5	Split Column by Example
2	2017-01-01 00:13:1	Duplicate Column
3	2017-01-01 00:16:1	Text Clustering
4	2017-01-01 00:21:2	Replace NA Values
5	2017-01-01 00:30:0	Handle Missing Values

2. For the example, enter a value of **Jan 01, 2017 12AM-2AM** for the first row.

IMPORTANT

In the previous example of deriving columns, you used multiple steps to derive a column that contained the date and time period. In this example, you can see that this operation can be performed as a single step by providing an example of the final output.

NOTE

You can give an example against any of the rows. For this example, the value of **Jan 01, 2017 12AM-2AM** is valid for the first row of data.

DERIVE COLUMN BY EXAMPLE: You have selected 1 source column and provided 0 examples. No suggestions Advanced mode			
#	starttime	abc Column	# start stati...
1	2017-01-01 00:06:58	Jan 01, 2017 12AM-2AM	67
2	2017-01-01 00:13:16	null	36
3	2017-01-01 00:16:17	null	36
4	2017-01-01 00:21:22	null	46

IMPORTANT

On a Mac, follow this step instead of step 3:

- Go to the first cell that contains **Jan 01, 2017 1AM-2AM**. It should be row 14. Correct the value to **Jan 01, 2017 12AM-2AM**, and select Enter.

3. Wait until the application computes the values against all the rows. The process might take several seconds. After the analysis is finished, use the **Review next suggested row** link to review data.

DERIVE COLUMN BY EXAMPLE: You have selected 1 source column and provided 1 example				Review next suggested row	Advanced mode
	<input checked="" type="checkbox"/> starttime	<input type="checkbox"/> abc Column	<input type="checkbox"/> # start stati...		
1	2017-01-01 00:06:58	Jan 01, 2017 12AM-2AM	67		
2	2017-01-01 00:13:16	Jan 01, 2017 12AM-2AM	36		
3	2017-01-01 00:16:17	Jan 01, 2017 12AM-2AM	36		
4	2017-01-01 00:21:22	Jan 01, 2017 12AM-2AM	46		

Ensure that the computed values are correct. If not, update the value with the expected value, and select Enter. Then wait for the analysis to finish. Complete the **Review next suggested row** process until you see **No suggestions**. **No suggestions** means the application looked at the edge cases and is satisfied with the synthesized program. It's a best practice to perform a visual inspection of the transformed data before you accept the transformation.

4. Select **OK** to accept the transform. Rename the newly created column to **Date Hour Range**.

	<input checked="" type="checkbox"/> starttime	<input type="checkbox"/> abc Date Hour Range	<input type="checkbox"/> # start stati...	
1	2017-01-01 00:06:58	Jan 01, 2017 12AM-2AM	67	
2	2017-01-01 00:13:16	Jan 01, 2017 12AM-2AM	36	
3	2017-01-01 00:16:17	Jan 01, 2017 12AM-2AM	36	
4	2017-01-01 00:21:22	Jan 01, 2017 12AM-2AM	46	

5. Right-click the **starttime** column header, and select **Remove column**.
6. To summarize the data, on the **Transform** menu, select **Summarize**. To create the transformation, use the following steps:
- Drag **Date Hour Range** and **start station id** to the **Group By** pane on the left.
 - Drag **start station id** to the **summarize data** pane on the right.

201701-hubway-tripdata

Group By

- Date Hour Ran.. ↑ ↓ ×
- start station id ↑ ↓ ×

Aggregate

- Count
- start station id

Column

New Column Name

startstation id_Count

OK Cancel

	abc Date Hour...	# start statio...			abc Date Hour...	# start statio...	# start stati...	
1	Jan 01, 2017 12..	67		▲	1	Jan 01, 2017 12..	67	▲
2	Jan 01, 2017 12..	36		▼	2	Jan 01, 2017 12..	36	▼
3	Jan 01, 2017 12..	36			3	Jan 01, 2017 12..	46	

7. Select **OK** to accept the summary result.

Join dataflows

To join the weather data with the trip data, use the following steps:

1. On the **Transforms** menu, select **Join**.
2. **Tables:** Select **BostonWeather** as the **Left** dataflow and **201701-hubway-tripdata** as the **Right** dataflow.
To continue, select **Next**.

Join

1. Tables
BostonWeather & 201701-hubway-tripdata

2. Key Columns
Select key columns

3. Join Type
Preview result

Left: BostonWeather

	abc Date Hour...	# N_DryBul...	# N_Relative...	# N_WindSp...
1	Jan 01, 2015 12..	0.29383886255...	0.42528735632...	0.28169014084...
2	Jan 01, 2015 2...	0.30331753554...	0.40229885057...	0.33802816901...
3	Jan 01, 2015 4...	0.29857819905...	0.42528735632...	0.45070422535...
4	Jan 01, 2015 6...	0.29857819905...	0.43678160919...	0.38028169014...
5	Jan 01, 2015 8...	0.33649289099...	0.32758620689...	0.42253521126...
6	Jan 01, 2015 10..	0.37440758293...	0.24712643678...	0.49295774647...
7	Jan 01, 2015 12..	0.39336492890...	0.22413793103...	0.42253521126...
8	Jan 01, 2015 2P..	0.38862559241...	0.24712643678...	0.40845070422...
9	Jan 01, 2015 4P..	0.37440758293...	0.30459770114...	0.42253521126...
10	Jan 01, 2015 6P..	0.37440758293...	0.31609195402...	0.39436619718...
11	Jan 01, 2015 8P..	0.37440758293...	0.39080459770...	0.30985915492...
12	Jan 01, 2015 10..	0.37914691943...	0.45977011494...	0.45070422535...
13	Jan 02, 2015 12..	0.39810426540...	0.41379310344...	0.33802816901...
14	Jan 02, 2015 2...	0.38862559241...	0.44827586206...	0.29577464788...

Right: 201701-hubway-tripdata

	abc Date Hour...	# start statio...	# startstatio...
1	Jan 01, 2017 12..	67	2
2	Jan 01, 2017 12..	36	2
3	Jan 01, 2017 12..	46	1
4	Jan 01, 2017 12..	10	1
5	Jan 01, 2017 12..	47	1
6	Jan 01, 2017 12..	107	1
7	Jan 01, 2017 12..	58	1
8	Jan 01, 2017 12..	9	1
9	Jan 01, 2017 12..	88	1
10	Jan 01, 2017 12..	89	1
11	Jan 01, 2017 12..	133	1
12	Jan 01, 2017 12..	27	1
13	Jan 01, 2017 12..	80	1
14	Jan 01, 2017 12..	22	1
15	Jan 01, 2017 12..	22	1

Previous **Next**

3. **Key Columns:** Select the **Date Hour Range** column in both the tables, and then select **Next**.

Join

1. Tables
BostonWeather & 201701-hubway-tripdata

2. Key Columns
Date Hour Range & Date Hour Range

3. Join Type
Preview result

Left: BostonWeather

abc	Date Hour...	# N_DryBulb...	# N_Relative...	# N_WindSp...
1	Jan 01, 2015 12..	0.29383886255...	0.42528735632...	0.28169014084...
2	Jan 01, 2015 2...	0.30331753554...	0.40229885057...	0.33802816901...
3	Jan 01, 2015 4...	0.29857819905...	0.42528735632...	0.45070422535...
4	Jan 01, 2015 6...	0.29857819905...	0.43678160919...	0.38028169014...
5	Jan 01, 2015 8...	0.33649289099...	0.32758620689...	0.42253521126...
6	Jan 01, 2015 10..	0.37440758293...	0.24712643678...	0.49295774647...
7	Jan 01, 2015 12..	0.39336492890...	0.22413793103...	0.42253521126...
8	Jan 01, 2015 2P..	0.38862559241...	0.24712643678...	0.40845070422...

Right: 201701-hubway-tripdata

abc	Date Hour...	# start statio...	# startstatio...
1	Jan 01, 2017 12..	67	2
2	Jan 01, 2017 12..	36	2
3	Jan 01, 2017 12..	46	1
4	Jan 01, 2017 12..	10	1
5	Jan 01, 2017 12..	47	1
6	Jan 01, 2017 12..	107	1
7	Jan 01, 2017 12..	58	1
8	Jan 01, 2017 12..	9	1

Previous **Next** **Finish**

4. Join Type: Select **Matching rows** as the join type, and then select **Finish**.

1. Tables
BostonWeather & 201701-hubway-tripdata

2. Key Columns
Date Hour Range & Hour Range

3. Join Type
Preview result

Choose the type of join you want
 Matching rows (8068)
 Unmatched rows from BostonWeather (9118)
 Unmatched rows from 201701-hubway-tripdata (0)

BostonWeather
9476 rows

Unmatched rows in BostonWeather
9118 rows

201701-hubway-tripdata
8068 rows

Matching rows
8068 rows

Result
8068 rows

Unmatched rows in 201701-hubway-tripdata
0 rows

Result: Join Result

abc	Date Hour...	# N_DryBulb...	# N_Relative...	# N_WindSp...	abc	Hour Range	# start statio...	# startstatio...
1	Jan 01, 2017 12..	0.45023696682...	0.87356321839...	0.43661971830...	Jan 01, 2017 12..	67	2	
2	Jan 01, 2017 12..	0.45023696682...	0.87356321839...	0.43661971830...	Jan 01, 2017 12..	36	2	
3	Jan 01, 2017 12..	0.45023696682...	0.87356321839...	0.43661971830...	Jan 01, 2017 12..	46	1	

This process creates a new dataflow named **Join Result**.

Create additional features

- To create a column that contains the day of the week, right-click the **Date Hour Range** column and select **Derive Column by Example**. Use a value of **Sun** for a date that occurred on a Sunday. An example is **Jan 01, 2017 12AM-2AM**. Select **Enter**, and then select **OK**. Rename this column to **Weekday**.

	<input checked="" type="checkbox"/> abc Hour Range	<input type="checkbox"/> abc Column	<input type="checkbox"/> # N_DryBulb...
1	Jan 01, 2017 12AM-2AM	Sun	0.45023696682...
2	Jan 01, 2017 12AM-2AM	Sun	0.45023696682...
3	Jan 01, 2017 12AM-2AM	Sun	0.45023696682...
4	Jan 01, 2017 12AM-2AM	Sun	0.45023696682...

2. To create a column that contains the time period for a row, right-click the **Date Hour Range** column, and select **Derive Column by example**. Use a value of **12AM-2AM** for the row that contains **Jan 01, 2017 12AM-2AM**. Select Enter, and then select **OK**. Rename this column to **Period**.

DERIVE COLUMN BY EXAMPLE: You have selected 1 source column and provided 1 example. Analyzing Data Advanced mode						
	<input checked="" type="checkbox"/> abc Hour Range	<input type="checkbox"/> abc Column	<input type="checkbox"/> abc Weekday	<input type="checkbox"/> # N_DryBulb...	<input type="checkbox"/> # N_RelativeHumidity	<input type="checkbox"/> # N_WindSpeed
1	Jan 01, 2017 12AM-2AM	12AM-2AM	Sun	0.45023696682...	0.8735632183908046	0.43661971830985913
2	Jan 01, 2017 12AM-2AM	12AM-2AM	Sun	0.45023696682...	0.8735632183908046	0.43661971830985913
3	Jan 01, 2017 12AM-2AM	12AM-2AM	Sun	0.45023696682...	0.8735632183908046	0.43661971830985913

3. To remove the **Date Hour Range** and **r_Date Hour Range** columns, select Ctrl (Command ⌘ on Mac), and then select each column header. Right-click, and select **Remove Column**.

Read data from Python

You can run a data preparation package from Python or PySpark and retrieve the result as a **Data Frame**.

To generate an example Python script, right-click **BikeShare Data Prep**, and select **Generate Data Access Code File**. The example Python file is created in your **Project Folder** and is also loaded in a tab within Workbench. The following Python script is an example of the code that is generated:

```
# Use the Azure Machine Learning data preparation package
from azureml.dataprep import package

# Use the Azure Machine Learning data collector to log various metrics
from azureml.logging import get_azureml_logger
logger = get_azureml_logger()

# This call will load the referenced package and return a DataFrame.
# If run in a PySpark environment, this call returns a
# Spark DataFrame. If not, it will return a Pandas DataFrame.
df = package.run('BikeShare Data Prep.dprep', dataflow_idx=0)

# Remove this line and add code that uses the DataFrame
df.head(10)
```

For this tutorial, the name of the file is `BikeShare Data Prep.py`. This file is used later in the tutorial.

Save test data as a CSV file

To save the **Join Result** dataflow to a .csv file, you must change the `BikeShare Data Prep.py` script.

1. Open the project for editing in Visual Studio Code.

File Edit Dataflows Transforms Inspectors View Help

Save Ctrl+S

Open Command Prompt

Open PowerShell

Open Project (VSCode)

Configure Project IDE

Proxy Manager

Quit Alt+F4

 Data Preparations (1)

 BikeShare Data Prep



Project Dashboard

Boston

Join Result

	abc Period	abc W
36	6AM-8AM	Sun
37	8AM-10AM	Sun
38	8AM-10AM	Sun
39	8AM-10AM	Sun
40	8AM-10AM	Sun
41	8AM-10AM	Sun

2. Update the Python script in the `BikeShare Data Prep.py` file by using the following code:

```

import pyspark

from azureml.dataprep.package import run
from pyspark.sql.functions import *

# start Spark session
spark = pyspark.sql.SparkSession.builder.appName('BikeShare').getOrCreate()

# dataflow_idx=2 sets the dataflow to the 3rd dataflow (the index starts at 0), the Join Result.
df = run('BikeShare Data Prep.dprep', dataflow_idx=2)
df.show(n=10)
row_count_first = df.count()

# Example file name: 'wasb://data-files@bikesharestorage.blob.core.windows.net/testata'
# 'wasb://<your container name>@<your azure storage name>.blob.core.windows.net/<csv folder name>
blobfolder = 'Your Azure Storage blob path'

df.write.csv(blobfolder, mode='overwrite')

# retrieve csv file parts into one data frame
csvfiles = "<Your Azure Storage blob path>/*.csv"
df = spark.read.option("header", "false").csv(csvfiles)
row_count_result = df.count()
print(row_count_result)
if (row_count_first == row_count_result):
    print('counts match')
else:
    print('counts do not match')
print('done')

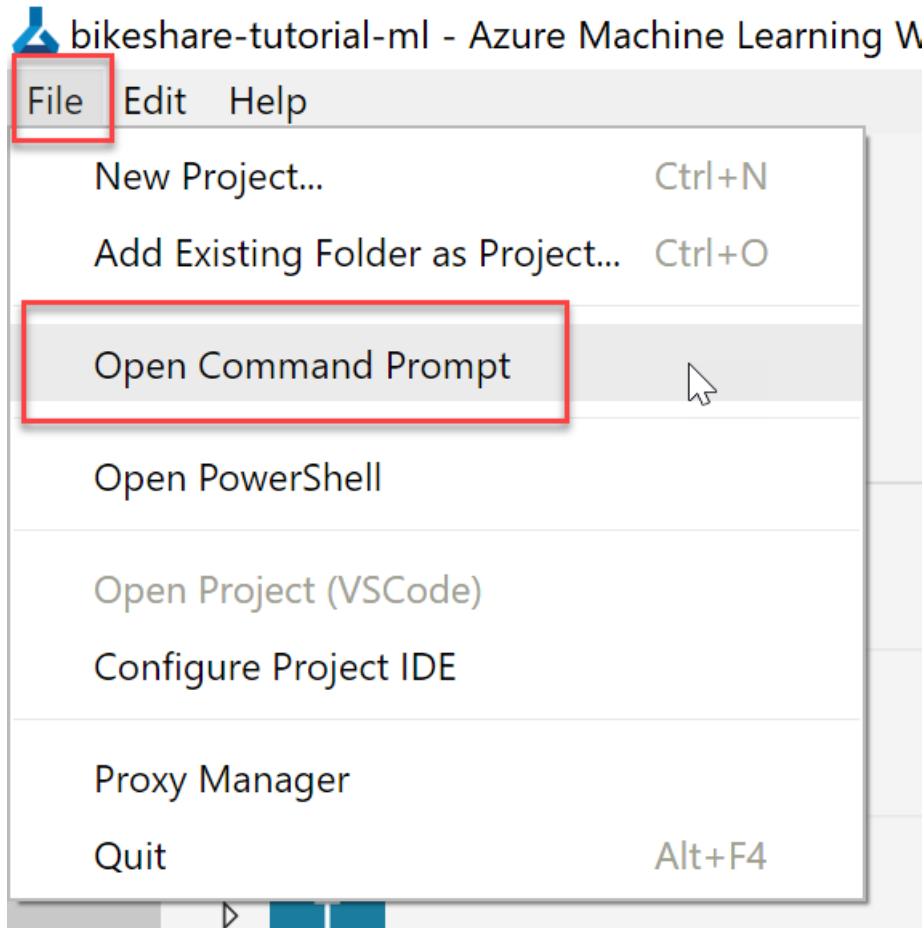
```

3. Replace `Your Azure Storage blob path` with the path to the output file to be created. Replace for both the `blobfolder` and `csvfiles` variables.

Create an HDInsight run configuration

1. In Machine Learning Workbench, open the command-line window, select the **File** menu, and then select **Open Command Prompt**. Your command prompt starts in the project folder with the prompt

```
C:\Projects\BikeShare> .
```



IMPORTANT

You must use the command-line window (opened from Workbench) to accomplish the steps that follow.

2. Use the command prompt to sign in to Azure.

The Workbench app and CLI use independent credential caches when you authenticate against Azure resources. You need to do this only once until the cached token expires. The `az account list` command returns the list of subscriptions available to your login. If there is more than one, use the ID value from the desired subscription. Set that subscription as the default account to use with the `az account set -s` command, and then provide the subscription ID value. Then confirm the setting by using the account `show` command.

```
REM login by using the aka.ms/devicelogin site
az login

REM lists all Azure subscriptions you have access to
az account list -o table

REM sets the current Azure subscription to the one you want to use
az account set -s <subscriptionId>

REM verifies that your current subscription is set correctly
az account show
```

3. Create the HDInsight run config. You need the name of your cluster and the `sshuser` password.

```
az ml computetarget attach cluster --name hdinsight --address <yourclustername>.azurehdinsight.net --
username sshuser --password <your password>
az ml experiment prepare -c hdinsight
```

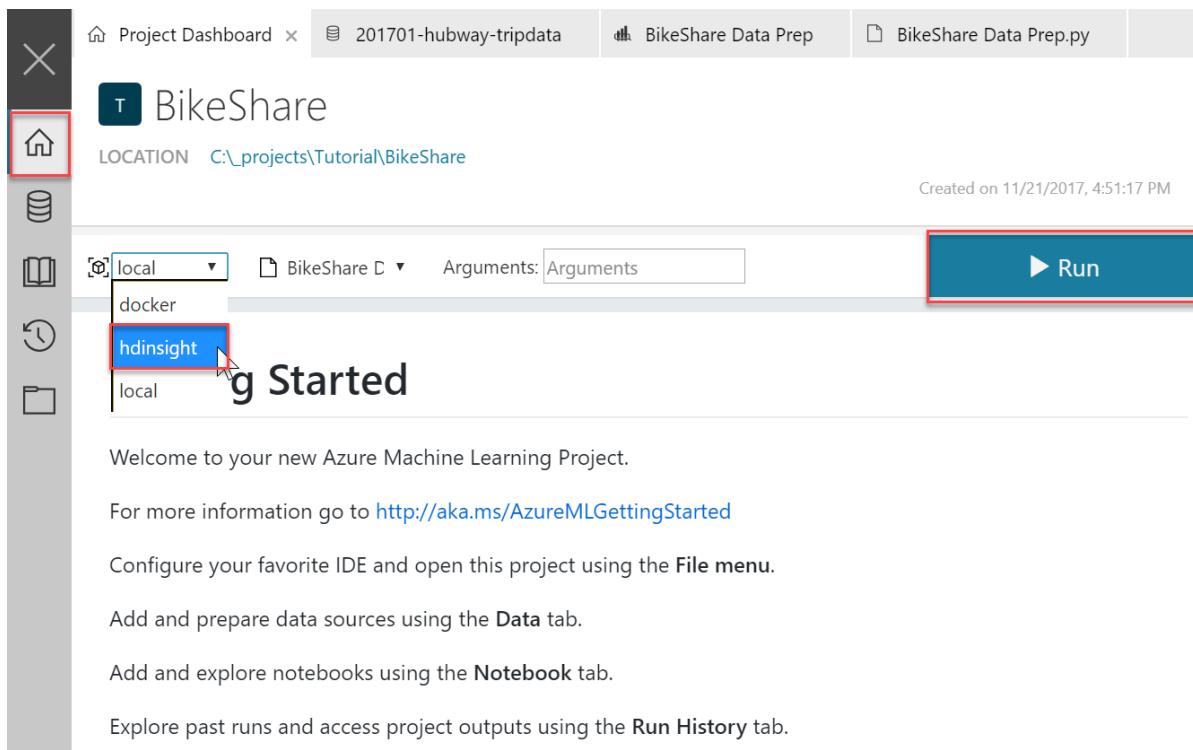
NOTE

When a blank project is created, the default run configurations are **local** and **docker**. This step creates a new run configuration that is available in Workbench when you run your scripts.

Run in an HDInsight cluster

Return to the Machine Learning Workbench application to run your script in the HDInsight cluster.

1. Return to the home screen of your project by selecting the **Home** icon on the left.
2. Select **hdinsight** from the drop-down list to run your script in the HDInsight cluster.
3. Select **Run**. The script is submitted as a job. The job status changes to **Completed** after the file is written to the specified location in your storage container.



Substitute data sources

In the previous steps, you used the `201701-hubway-tripdata.csv` and `BostonWeather.csv` data sources to prepare the test data. To use the package with the other trip data files, use the following steps:

1. Create a new data source by using the steps given previously, with the following changes to the process:

- **File Selection:** When you select a file, multi-select the six remaining trip tripdata .csv files.

Browse to find the file

Browse to the path of the file you would like to use.

Path

Azure Blob ▾

<https://bikesharestorage.blob.core.windows.net/data-files/tripdata/2015>

+5

[Browse...](#)

NOTE

The +5 entry indicates that there are five additional files beyond the one that is listed.

- **File Details:** Set **Promote Headers Mode** to **All Files Have The Same Headers**. This value indicates that each of the files contains the same header.

Choose file parameters

Set parameters to interpret the file.

File Type

Delimited File (csv, tsv, txt, etc.)

Comma [,]

Skip Lines Mode

Don't skip

File Encoding

utf-8

Promote Headers Mode

All Files Have The Same Headers

Handle Quoted Line Breaks? (will increase run times in Spark)

Save the name of this data source because it's used in later steps.

2. Select the folder icon to view the files in your project. Expand the **aml_config** directory, and then select the `hdinsight.runconfig` file.

The screenshot shows the Azure Data Studio interface. On the left is the file explorer with a tree view of files and folders. A red box highlights the 'hdinsight.runconfig' file under the 'Tutorial' folder. On the right is the code editor window titled 'hdinsight.runconfig'. The code contains configuration settings for a PySpark job. A red box highlights the 'DataSourceSubstitutions' section, which includes the line '201701-hubway-tripdata.dsor...'. The status bar at the bottom of the code editor shows 'Arguments: Arguments'.

```
ArgumentVector:  
- $file  
CondaDependenciesFile: aml_config/conda_dependencies.yml  
EnvironmentVariables: null  
Framework: PySpark  
PrepareEnvironment: false  
SparkDependenciesFile: aml_config/spark_dependencies.yml  
Target: hdinsight  
TrackedRun: true  
UseSampling: true  
  
DataSourceSubstitutions:  
201701-hubway-tripdata.dsor...
```

3. Select the **Edit** button to open the file in Visual Studio Code.
4. Add the following lines at the end of the `hdinsight.runconfig` file, and then select the disk icon to save the file.

```
DataSourceSubstitutions:  
201701-hubway-tripdata.dsor...
```

This change replaces the original data source with the one that contains the six trip data files.

Save training data as a CSV file

1. Browse to the Python file `BikeShare Data Prep.py` that you edited previously. Provide a different file path to save the training data.

```

import pyspark

from azureml.dataprep.package import run
from pyspark.sql.functions import *

# start Spark session
spark = pyspark.sql.SparkSession.builder.appName('BikeShare').getOrCreate()

# dataflow_idx=2 sets the dataflow to the 3rd dataflow (the index starts at 0), the Join Result.
df = run('BikeShare Data Prep.dprep', dataflow_idx=2)
df.show(n=10)
row_count_first = df.count()

# Example file name: 'wasb://data-files@bikesharestorage.blob.core.windows.net/traindata'
# 'wasb://<your container name>@<your azure storage name>.blob.core.windows.net/<csv folder name>
blobfolder = 'Your Azure Storage blob path'

df.write.csv(blobfolder, mode='overwrite')

# retrieve csv file parts into one data frame
csvfiles = "<Your Azure Storage blob path>/*.csv"
df = spark.read.option("header", "false").csv(csvfiles)
row_count_result = df.count()
print(row_count_result)
if (row_count_first == row_count_result):
    print('counts match')
else:
    print('counts do not match')
print('done')

```

2. Use the folder named `traindata` for the training data output.
3. To submit a new job, select **Run**. Make sure **hdinsight** is selected. A job is submitted with the new configuration. The output of this job is the training data. This data is created by using the same data preparation steps that you followed previously. The job might take a few minutes to finish.

Clean up resources

IMPORTANT

The resources you created can be used as prerequisites to other Azure Machine Learning service tutorials and how-to articles.

If you don't plan to use the resources you created here, delete them so you don't incur any charges.

1. In the Azure portal, select **Resource groups** on the far left.

The screenshot shows the Microsoft Azure portal interface. The left sidebar has a 'Resource groups' item selected, which is highlighted with a red box. The main content area shows the 'newacct' resource group details. At the top of the main area, there is a toolbar with various icons and a 'Delete resource group' button, which is also highlighted with a red box. Below the toolbar, there is an 'Essentials' section and a table listing one item: 'newacct' (Azure Cosmos DB account) located in South Central US.

NAME	TYPE	LOCATION
newacct	Azure Cosmos DB account	South Central US

2. From the list, select the resource group you created.
3. Select **Delete resource group**.
4. Enter the resource group name, and then select **Delete**.

Next steps

You have finished the bike-share data preparation tutorial. In this tutorial, you used Machine Learning (preview) to learn how to:

- Prepare data interactively with the Machine Learning data preparation tool.
- Import, transform, and create a test dataset.
- Generate a data preparation package.
- Run the data preparation package by using Python.
- Generate a training dataset by reusing the data preparation package for additional input files.

Next, learn more about data preparation:

[Data preparation user guide](#)

Azure Machine Learning Model Management

12/11/2018 • 5 minutes to read • [Edit Online](#)

NOTE

This article is deprecated. Support for earlier versions of this service will end incrementally. [View the support timeline](#). Start using the latest version with this [quickstart](#).

Azure Machine Learning Model Management enables you to manage and deploy machine-learning workflows and models.

Model Management provides capabilities for:

- Model versioning
- Tracking models in production
- Deploying models to production through AzureML Compute Environment with [Azure Container Service](#) and [Kubernetes](#)
- Creating Docker containers with the models and testing them locally
- Automated model retraining
- Capturing model telemetry for actionable insights.

Azure Machine Learning Model Management provides a registry of model versions. It also provides automated workflows for packaging and deploying Machine Learning containers as REST APIs. The models and their runtime dependencies are packaged in Linux-based Docker container with prediction API.

Azure Machine Learning Compute Environments help to set up and manage scalable clusters for hosting the models. The compute environment is based on Azure Container Services. Azure Container Services provides automatic exposure of Machine Learning APIs as REST API endpoints with the following features:

- Authentication
- Load balancing
- Automatic scale-out
- Encryption

Azure Machine Learning Model Management provides these capabilities through the CLI, API, and the Azure portal.

Azure Machine Learning model management uses the following information:

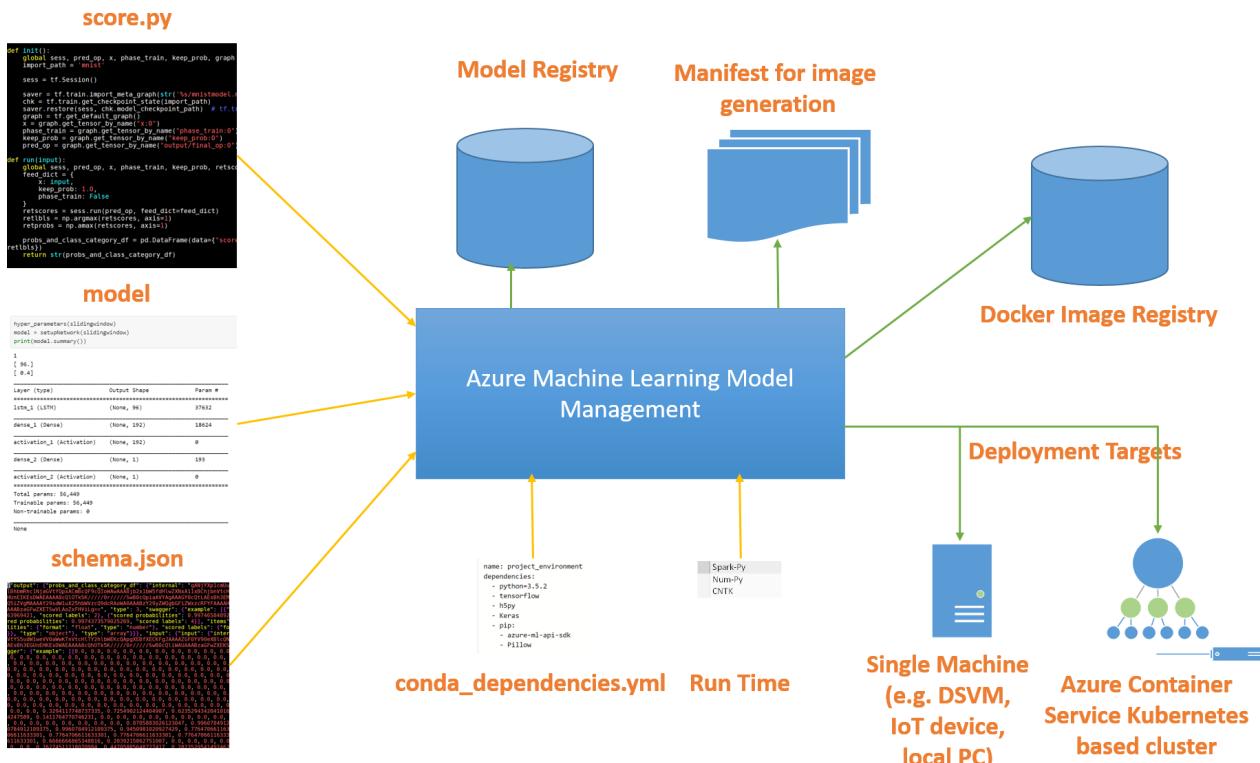
- Model file or a directory with the model files
- User created Python file implementing a model scoring function
- Conda dependency file listing runtime dependencies
- Runtime environment choice, and
- Schema file for API parameters

This information is used when performing the following actions:

- Registering a model
- Creating a manifest that is used when building a container
- Building a Docker container image

- Deploying a container to Azure Container Service

The following figure shows an overview of how models are registered and deployed into the cluster.



Create and manage models

You can register models with Azure Machine Learning Model Management for tracking model versions in production. For ease of reproducibility and governance, the service captures all dependencies and associated information. For deeper insights into performance, you can capture model telemetry using the provided SDK. Model telemetry is archived in user-provided storage. The model telemetry can be used later for analyzing model performance, retraining, and gaining insights for your business.

Create and manage manifests

Models require additional artifacts to deploy into production. The system provides the capability to create a manifest that encompasses model, dependencies, inference script (aka scoring script), sample data, schema etc. This manifest acts as a recipe to create a Docker container image. Enterprises can auto-generate manifest, create different versions, and manage their manifests.

Create and manage Docker container images

You can use the manifest from the previous step to build Docker-based container images in their respective environments. The containerized, Docker-based images provide enterprises with the flexibility to run these images on the following compute environments:

- Kubernetes based Azure Container Service
 - On-premises container services
 - Development environments
 - IoT devices

These Docker-based containerized images are self-contained with all necessary dependencies required for generating predictions.

Deploy Docker container images

With the Azure Machine Learning Model Management, you can deploy Docker-based container images with a single command to Azure Container Service managed by ML Compute Environment. These deployments are created with a front-end server that provides the following features:

- Low latency predictions at scale
- Load balancing
- Automatic scaling of ML endpoints
- API key authorization
- API swagger document

You can control the deployment scale and telemetry through the following configuration settings:

- System logging and model telemetry for each web service level. If enabled, all stdout logs are streamed to [Azure Application Insights](#). Model telemetry is archived in storage that you provide.
- Auto-scale and concurrency limits. These settings automatically increase the number of deployed containers based on the load within the existing cluster. They also control the throughput and consistency of prediction latency.

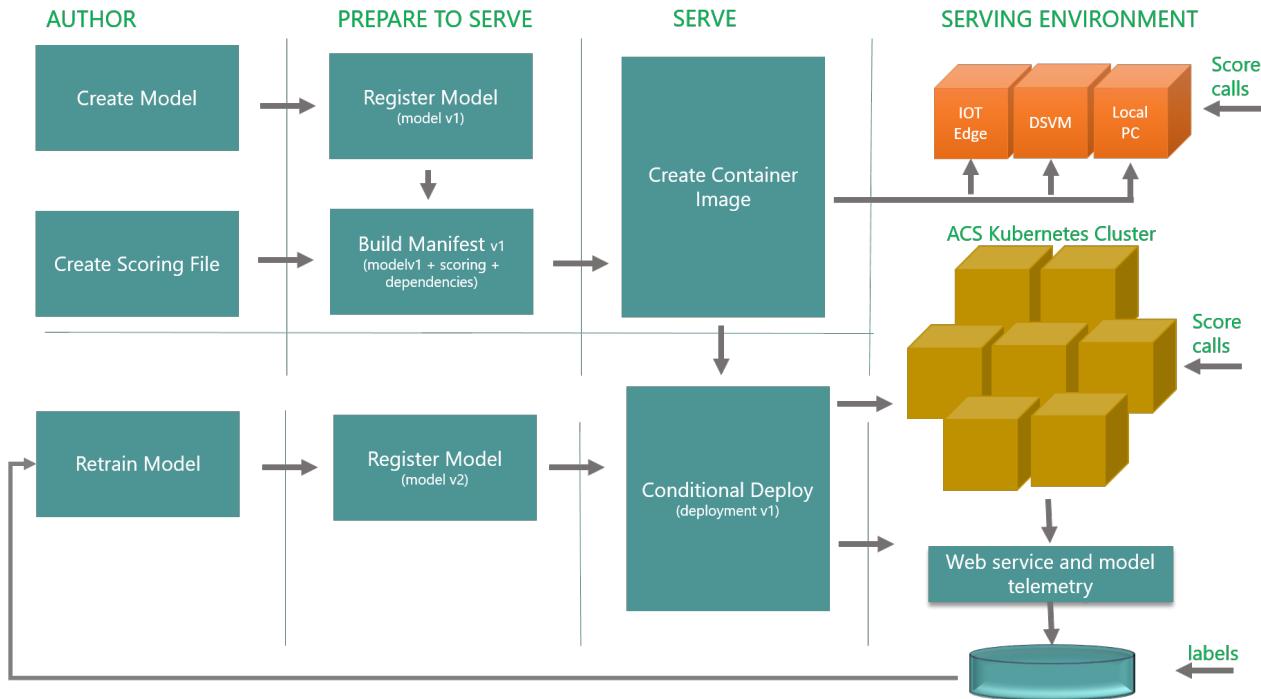
Consumption

Azure Machine Learning Model Management creates REST API for the deployed model along with the swagger document. You can consume deployed models by calling the REST APIs with API key and model inputs to get the predictions as part of the line-of-business applications. The sample code is available in GitHub for languages Java, [Python](#), and C# for calling REST APIs. The Azure Machine Learning Model Management CLI provides an easy way to work with these REST APIs. You can consume the APIs using a single CLI command, a swagger-enabled applications, or using curl.

Retraining

Azure Machine Learning Model Management provides APIs that you can use to retrain your models. You can also use the APIs to update existing deployments with updated versions of the model. As part of the data science workflow, you recreate the model in your experimentation environment. Then, you register the model with Model Management, and update existing deployments. Updates are performed using a single UPDATE CLI command. The UPDATE command updates existing deployments without changing the API URL or the key. The applications consuming the model continue to work without any code change, and start getting better predictions using new model.

The complete workflow describing these concepts is captured in the following figure:



Frequently asked questions (FAQ)

- **What data types are supported? Can I pass NumPy arrays directly as input to web service?**

If you are providing schema file that was created using generate_schema SDK, then you can pass NumPy and/or Pandas DF. You can also pass any JSON serializable inputs. You can pass image as binary encoded string as well.

- **Does the web service support multiple inputs or parse different inputs?**

Yes, you can take multiple inputs packaged in the one JSON request as a dictionary. Each input would correspond to a single unique dictionary key.

- **Is the call activated by a request to the web service a blocking call or an asynchronous call?**

If service was created using realtime option as part of the CLI or API, then it is a blocking/synchronous call. It is expected to be realtime fast. Although on the client side you can call it using async HTTP library to avoid blocking the client thread.

- **How many requests can the web service simultaneously handle?**

It depends on the cluster and web service scale. You can scale out your service to 100x of replicas and then it can handle many requests concurrently. You can also configure the maximum concurrent request per replica to increase service throughput.

- **How many requests can the web service queue up?**

It is configurable. By default, it is set to ~10 per single replica, but you can increase/decrease it to your application requirements. Typically, increasing it the number of queued requests increases the service throughput but makes the latencies worse at higher percentiles. To keep the latencies consistent, you may want to set the queuing to a low value (1-5), and increase the number of replicas to handle the throughput. You can also turn on autoscaling to make the number of replicas adjusting automatically based on load.

- **Can the same machine or cluster be used for multiple web service endpoints?**

Absolutely. You can run 100x of services/endpoints on the same cluster.

Next steps

For getting started with Model Management, see [Configuring Model Management](#).

Install and use the machine learning CLI for top tasks in Azure Machine Learning

9/24/2018 • 4 minutes to read • [Edit Online](#)

NOTE

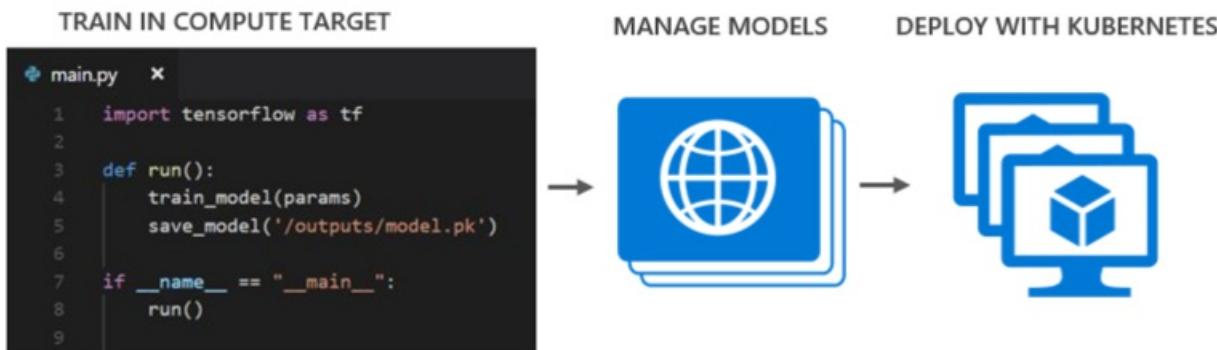
This article is deprecated. Support for earlier versions of this service will end incrementally. [View the support timeline](#). Start using the latest version with this [quickstart](#).

Azure Machine Learning service is an integrated, end-to-end data science and advanced analytics solution. Professional data scientists can use Azure Machine Learning service to prepare data, develop experiments, and deploy models at cloud scale.

Azure Machine Learning provides a command-line interface (CLI) with which you can:

- Manage your workspace and projects
- Set up compute targets
- Run training experiments
- View history and metrics for past runs
- Deploy models into production as web services
- Manage production models and services

This article presents some of the most useful CLI commands for your convenience.



What you need to get started

You need contributor access to an Azure subscription or a resource group where you can deploy your models. Also, you need to install Azure Machine Learning Workbench in order to run the CLI.

IMPORTANT

The CLI delivered with Azure Machine Learning service is different from the [Azure CLI](#), which is used for managing Azure resources.

Get and start CLI

To get this CLI, install Azure Machine Learning Workbench, which can be downloaded from here:

- Windows - <https://aka.ms/azureml-wb-msi>
- MacOS - <https://aka.ms/azureml-wb-dmg>

To start the CLI:

- In Azure Machine Learning Workbench, launch the CLI from the menu **File -> Open Command Prompt**.

Get command help

You can get extra information about CLI commands using `--debug` or `--help` after the commands such as `az ml <xyz> --debug` where `<xyz>` is the command name. For example:

```
az ml computetarget --debug

az ml experiment --help
```

Common CLI tasks for Azure Machine Learning

Learn about the most common tasks you can perform with the CLI in this section, including:

- [Setting up compute targets](#)
- [Submitting jobs for remote execution](#)
- [Working with Jupyter notebooks](#)
- [Interacting with run histories](#)
- [Configuring your environment to operationalize](#)

Set up a compute target

You can compute your machine learning model in different targets, including:

- in local mode
- in a Data Science VM (DSVM)
- on an HDInsight cluster

To attach a Data Science VM target:

```
az ml computetarget attach remotedocker -n <target name> -a <ip address or FQDN> -u <username> -w <password>
```

To attach an HDInsight target:

```
az ml computetarget attach cluster -n <target name> -a <cluster name, e.g. myhdicluster-ssh.azurehdinsight.net>
-u <ssh username> -w <ssh password>
```

Within the **aml_config** folder, you can change the conda dependencies.

Also, you can operate with PySpark, Python, or Python in a GPU DSVM.

To define the Python operation mode:

- For Python, add `Framework:Python` in `<target name>.runconfig`
- For PySpark, add `Framework:PySpark` in `<target name>.runconfig`
- For Python in a GPU DSVM,
 1. Add `Framework:Python` in `<target name>.runconfig`

2. Also, add `baseDockerImage: microsoft/mmlspark:plus-gpu-0.9.9` and `nvidiaDocker:true` in
`<target name>.compute`

To prepare the compute target:

```
az ml experiment prepare -c <target name>
```

TIP

To show your subscription:

```
az account show
```

To set your subscription:

```
az account set -s "my subscription name"
```

Submit remote jobs

To submit a job to a remote target:

```
az ml experiment submit -c <target name> myscript.py
```

Work with Jupyter notebooks

To start a Jupyter notebook:

```
az ml notebook start
```

This command starts a Jupyter notebook in localhost. You can work in local by selecting the kernel Python 3, or work in your remote VM by selecting the kernel `<target name>`.

Interact with and explore the run history

To list the run history:

```
az ml history list -o table
```

To list all completed runs:

```
az ml history list --query "[?status=='Completed']" -o table
```

To find runs with the best accuracy:

```
az ml history list --query "@[?Accuracy != null] | max_by(@, &Accuracy).Accuracy"
```

You can also download the files generated by each run.

To promote a model that is saved in the folder outputs:

```
az ml history promote -r <run id> -ap outputs/model.pkl -n <model name>
```

To download that model:

```
az ml asset download -l assets/model.pkl.link -d <model folder path>
```

Configure your environment to operationalize

To set up your operationalization environment, you must create:

- A resource group
- A storage account
- An Azure Container Registry (ACR)
- An Application insight account
- A Kubernetes deployment on an Azure Container Service (ACS) cluster

To set up a local deployment for testing in a Docker container:

```
az ml env setup -l <region, e.g. eastus2> -n <env name> -g <resource group name>
```

To set up an ACS cluster with Kubernetes:

```
az ml env setup -l <region, e.g. eastus2> -n <env name> -g <resource group name> --cluster
```

To monitor the status of the deployment:

```
az ml env show -n <environment name> -g <resource group name>
```

To set the environment that should be used:

```
az ml env set -n <environment name> -g <resource group name>
```

Next steps

Get started with one of these articles:

- [Install and start using Azure Machine Learning](#)
- [Classifying Iris Data Tutorial: Part 1](#)

Dig deeper with one of these articles:

- [CLI commands for managing models](#)

How to configure Azure Machine Learning Workbench to work with an IDE

9/24/2018 • 2 minutes to read • [Edit Online](#)

NOTE

This article is deprecated. Support for earlier versions of this service will end incrementally. [View the support timeline](#). Start using the latest version with this [quickstart](#).

Azure Machine Learning Workbench can be configured to work with popular Python IDEs (Integrated Development Environment). It enables a smooth data science development experience moving between data preparation, code authoring, run tracking and operationalization. Currently the supported IDEs are:

- Microsoft Visual Studio Code
- JetBrains PyCharm

Configure workbench

1. Click on the **File** menu in the top left corner of the app.
2. Select the **Configure Project IDE** option from the flyout
3. Type in `VS Code` or `PyCharm` in the **Name** field (the name is arbitrary)
4. Enter the location to the IDE executable (complete with the executable name and extension) in **Execution Path**

Default install path for Visual Studio Code

- Windows 32-bit - `C:\Program Files (x86)\Microsoft VS Code\Code.exe`
- Windows 64-bit - `C:\Program Files\Microsoft VS Code\Code.exe`
- macOS - select the .app path, for example `/Applications/Visual Studio Code.app`, and the app appends the rest of the path for you. The full path to the executable by default is
`/Applications/Visual Studio Code.app/Contents/Resources/app/bin/code`. If you have executed the
Shell Command: Install 'code' command in PATH command in VS Code, then you can also reference the VS Code
script at `/usr/local/bin/code`

Default install path for PyCharm

- Windows 32-bit - `C:\Program Files (x86)\JetBrains\PyCharm Community Edition 2017.2.1\bin\pycharm.exe`
- Windows 64-bit - `C:\Program Files\JetBrains\PyCharm Community Edition 2017.2.1\bin\pycharm64.exe`
- macOS - select the .app path, for example `/Applications/PyCharm CE.app`, and the app appends the rest of the path for you. The full path to the executable by default is `/Applications/PyCharm CE.app/Contents/MacOS/pycharm`.
You may also find PyCharm at the bin folder, `/usr/local/bin/charm`

Open project in IDE

Once the configuration is complete, you can open an Azure Machine Learning project by opening the **File** menu in Azure Machine Learning Workbench, then click **Open Project (<IDE_Name>)**. This action opens the current active project in the configured IDE. *Note: If you are not in a project, the **Open Project (<IDE_Name>)** will be disabled.*

Configuring the integrated terminal in Visual Studio Code

Windows

We have overridden the default shell to be cmd instead of PowerShell. On clicking on **Open Project (<IDE_Name>)**, you see a prompt:

Do you allow shell: C:\windows\System32\cmd.exe (defined as a workspace setting) to be launched in the terminal?

Answer `yes` to allow configuring the shell to work seamlessly with Azure ML Workbench command-line interface.

Mac

To run an `az` command using Visual Studio Code's integrated terminal on Mac, you need to manually set the `PATH` to be the same value as `PATH` in the project's `.vscode/settings.json` file under the key `terminal.integrated.env.osx`. You can do so by running the following command in the terminal:
`PATH=<PATH in .vscode/settings>`

Use Jupyter Notebooks in Azure Machine Learning Workbench

9/24/2018 • 5 minutes to read • [Edit Online](#)

NOTE

This article is deprecated. Support for earlier versions of this service will end incrementally. [View the support timeline](#). Start using the latest version with this [quickstart](#).

Azure Machine Learning Workbench supports interactive data science experimentation through its integration with Jupyter Notebooks. This article describes how to make effective use of this feature to increase the rate and quality of your interactive data science experimentation.

Prerequisites

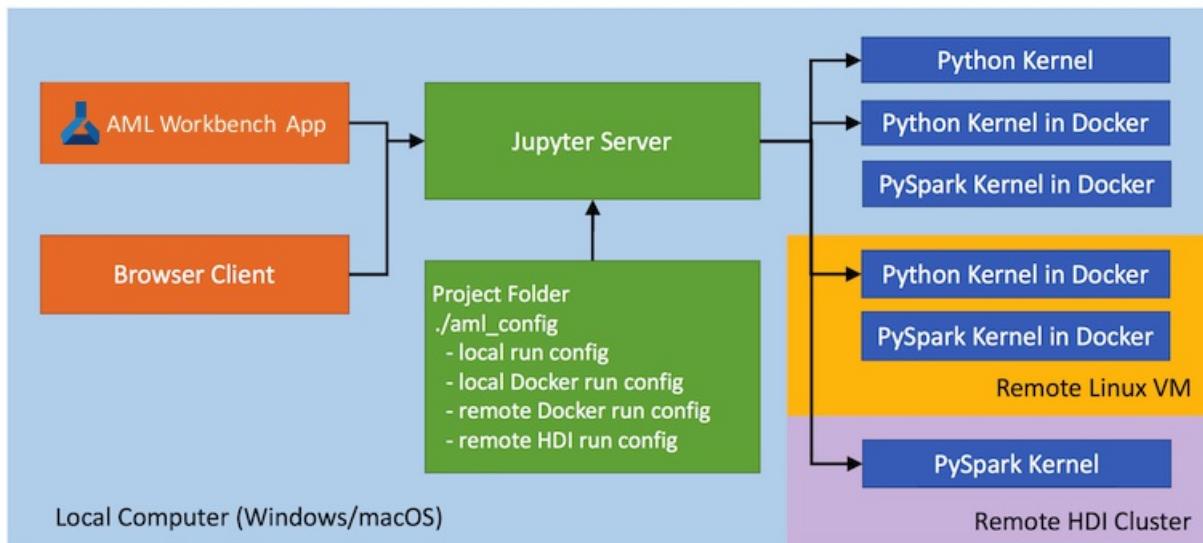
- [Create Azure Machine Learning accounts and install Azure Machine Learning Workbench](#).
- Be familiar with the [Jupyter Notebook](#). This article is not about learning how to use Jupyter.

Jupyter Notebook architecture

At a high level, Jupyter Notebook architecture includes three components. Each can run in different compute environments:

- **Client:** Receives user input and displays rendered output.
- **Server:** Web server that hosts the notebook files (.ipynb files).
- **Kernel:** Runtime environment in which execution of the notebook cells happens.

For more information, see the official [Jupyter documentation](#). The following diagram depicts how this client, server, and kernel architecture maps to the components in Azure Machine Learning:



Kernels in Azure Machine Learning Workbench notebooks

You can access different kernels in Azure Machine Learning Workbench by defining run configurations and

compute targets in the `aml_config` folder in your project. Adding a new compute target by issuing the `az ml computetarget attach` command is the equivalent of adding a new kernel.

NOTE

Review [Configuring Azure Machine Learning Experimentation Service](#) for more details on run configurations and compute targets.

Kernel naming convention

Azure Machine Learning Workbench generates custom Jupyter kernels. These kernels are named `<project name><run config name>`. For example, if you have a run configuration named `docker-python` in a project named `myIris`, Azure Machine Learning makes available a kernel named `myIris docker-python`. You set the running kernel in the Jupyter Notebook **Kernel** menu, in the **Change kernel** submenu. The name of the running kernel appears on the far right of the menu bar.

Currently, Azure Machine Learning Workbench supports the following types of kernels.

Local Python kernel

This Python kernel supports execution on local machines. It's integrated with Azure Machine Learning Run History support. The name of the kernel is typically `my_project_name local`.

NOTE

Do not use the Python 3 kernel. It is a standalone kernel provided by Jupyter by default and is not integrated with Azure Machine Learning capabilities. For example, the `%azureml` Jupyter magic functions return "not found" errors.

Python kernel in Docker (local or remote)

This Python kernel runs in a Docker container either on your local machine or in a remote Linux virtual machine (VM). The name of the kernel is typically `my_project docker`. The associated `docker.runconfig` file has the `Framework` field set to `Python`.

PySpark kernel in Docker (local or remote)

This PySpark kernel executes scripts in a Spark context running inside a Docker container, either on your local machine or on a remote Linux VM. The kernel name is typically `my_project docker`. The associated `docker.runconfig` file has the `Framework` field set to `PySpark`.

PySpark kernel in an Azure HDInsight cluster

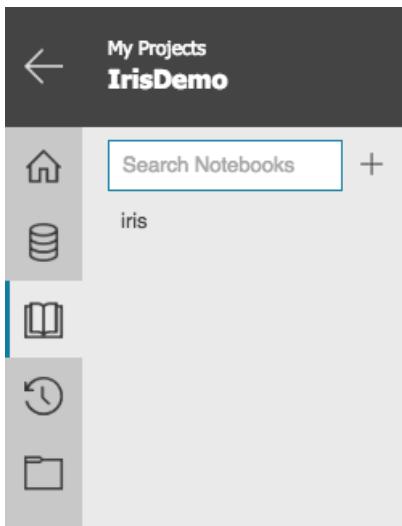
This kernel runs in the remote Azure HDInsight cluster that you attached as a compute target for your project. The kernel name is typically `my_project my_hdi`.

IMPORTANT

In the `.compute` file for the HDI compute target, you must change the `yarnDeployMode` field to `client` (the default value is `cluster`) to use this kernel.

Start a Jupyter server from Azure Machine Learning Workbench

From Azure Machine Learning Workbench, you can access notebooks via the **Notebooks** tab. The *Classifying Iris* sample project includes an `iris.ipynb` sample notebook.



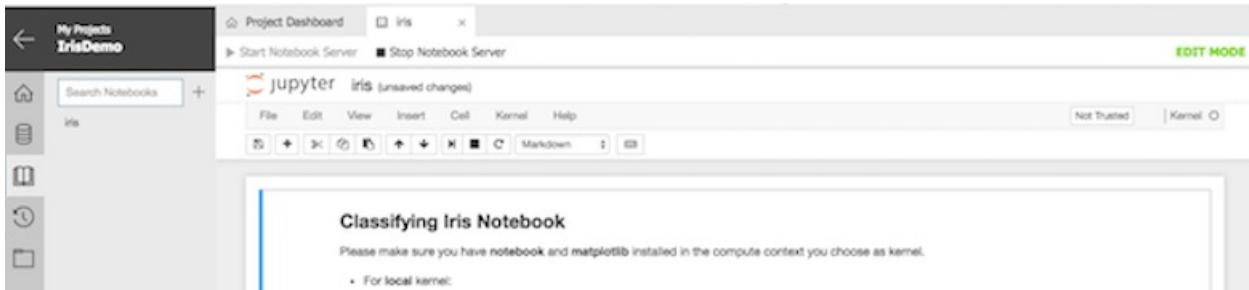
When you open a notebook in Azure Machine Learning Workbench, it's displayed in its own document tab in **Preview Mode**. This is a read-only view that doesn't require a running Jupyter server and kernel.



Selecting the **Start Notebook Server** button starts the Jupyter server and switches the notebook into **Edit Mode**. The familiar Jupyter Notebook user interface appears embedded in Workbench. You can now set a kernel from the **Kernel** menu and start your interactive notebook session.

NOTE

With non-local kernels, it can take a minute or two to start if you're using it for the first time. You can execute the `az ml experiment prepare` command from the CLI window to prepare the compute target so the kernel starts much faster after the compute target is prepared.



This is a fully interactive Jupyter Notebook experience. All regular notebook operations and keyboard shortcuts are supported from this window, except for some file operations that can be done via the Workbench **Notebooks** tab and **File** tab.

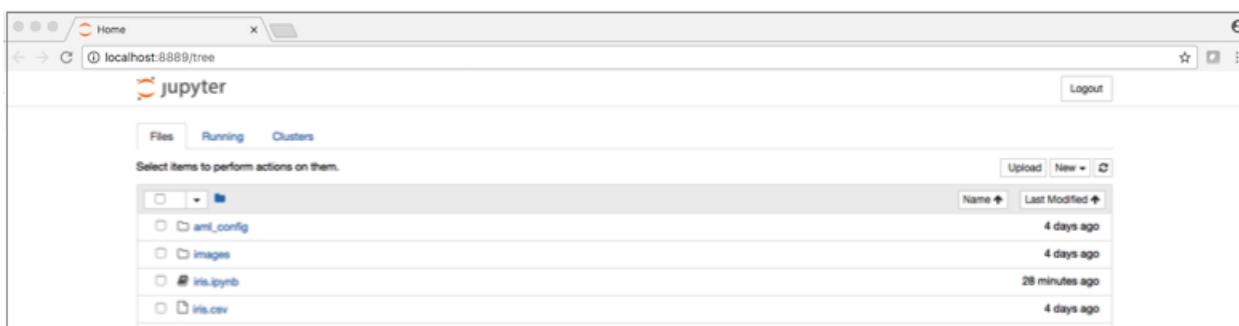
Start a Jupyter server from the command line

You can also start a notebook session by issuing `az ml notebook start` from the command-line window:

```
$ az ml notebook start
[I 10:14:25.455 NotebookApp] The port 8888 is already in use, trying another port.
[I 10:14:25.464 NotebookApp] Serving notebooks from local directory: /Users/johnpelak/Desktop/IrisDemo
[I 10:14:25.465 NotebookApp] 0 active kernels
[I 10:14:25.465 NotebookApp] The Jupyter Notebook is running at: http://localhost:8889/?token=1f0161ab88b22fc83f2083a93879ec5e8d0ec18490f0b953
[I 10:14:25.465 NotebookApp] Use Control-C to stop this server and shut down all kernels (twice to skip confirmation).
[C 10:14:25.466 NotebookApp]

Copy and paste this URL into your browser when you connect for the first time, to login with a token:
http://localhost:8889/?token=1f0161ab88b22fc83f2083a93879ec5e8d0ec18490f0b953
[I 10:14:25.759 NotebookApp] Accepting one-time-token-authenticated connection from ::1
[I 10:16:52.970 NotebookApp] Kernel started: 7f8932e0-89b9-48b4-b5d0-e8f48d1da159
[I 10:16:53.854 NotebookApp] Adapting to protocol v5.1 for kernel 7f8932e0-89b9-48b4-b5d0-e8f48d1da159
```

Your default browser automatically opens with the Jupyter server pointing to the project home directory. You can also use the URL and token displayed in the CLI window to open other browser windows locally.



You can now select an `.ipynb` notebook file, open it, set the kernel (if it hasn't been set), and start your interactive session.



Use magic commands to manage experiments

You can use [magic commands](#) within your notebook cells to track your run history and save outputs such as models or datasets.

To track individual notebook cell runs, use the `%azureml history on` magic command. After you turn on the history, each cell run appears as an entry in the run history:

```
%azureml history on
from azureml.logging import get_azureml_logger
logger = get_azureml_logger()
logger.log("Cell","Load Data")
```

To turn off cell run tracking, use the `%azureml history off` magic command.

You can use the `%azureml upload` magic command to save model and data files from your run. The saved objects appear as outputs in the run history view:

```
modelpath = os.path.join("outputs","model.pkl")
with open(modelpath,"wb") as f:
    pickle.dump(model,f)
%azureml upload outputs/model.pkl
```

NOTE

The outputs must be saved to a folder named *outputs*.

Next steps

- To learn how to use Jupyter Notebook, see the [Jupyter official documentation](#).
- To gain a deeper understanding of the Azure Machine Learning experimentation execution environment, see [Configuring Azure Machine Learning Experimentation Service](#).

Use a Git repo with a Machine Learning Workbench project

9/24/2018 • 8 minutes to read • [Edit Online](#)

NOTE

This article is deprecated. Support for earlier versions of this service will end incrementally. [View the support timeline](#). Start using the latest version with this [quickstart](#).

Learn how Azure Machine Learning Workbench uses Git to provide version control, and ensure reproducibility in your data science experiment. Learn how to associate your project with a cloud Git repository (repo).

Machine Learning Workbench is designed for Git integration. When you create a new project, the project folder is automatically "Git-initialized" as a local Git repo. A second, hidden local Git repo is also created, with a branch named AzureMLHistory/<project GUID>. The branch keeps track of project folder changes for each execution.

Associating the Azure Machine Learning project with a Git repo enables automatic version control, locally and remotely. The Git repo is hosted in Azure DevOps. Because the Machine Learning project is associated with a Git repo, anybody who has access to the remote repo can download the latest source code to another computer (roaming).

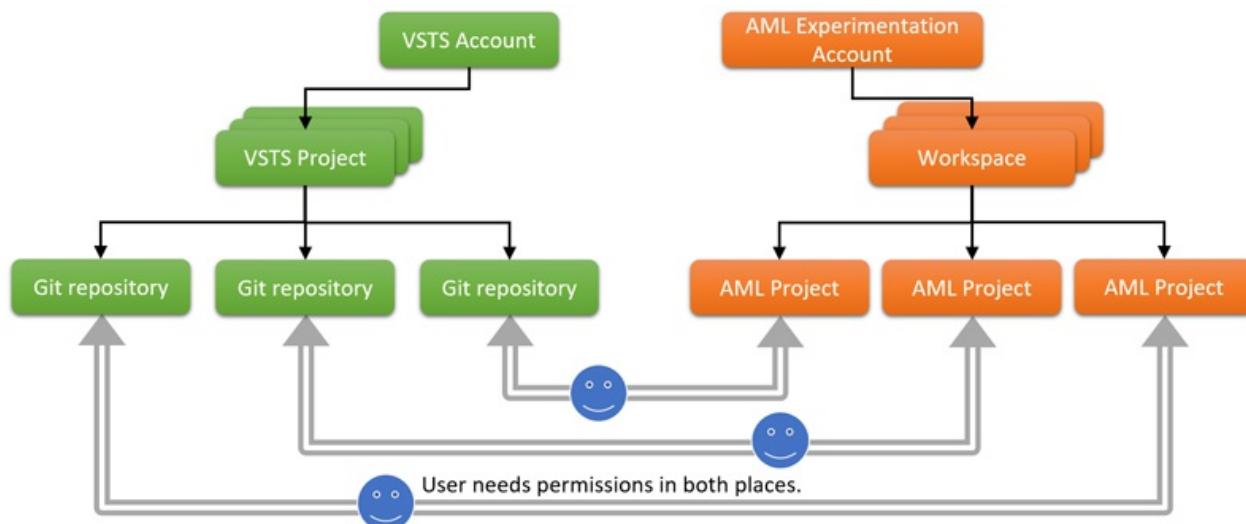
NOTE

Azure DevOps has its own access control list (ACL), which is independent of the Azure Machine Learning Experimentation service. User access might vary between a Git repo and a Machine Learning workspace or project. You might need to manage access.

Whether you want to give a team member code-level access to your Machine Learning project or just share the workspace, you need to grant the user the correct permissions to access the Azure DevOps Git repo.

To manage version control with Git, you can either use the master branch or create other branches in the repo. You can also use the local Git repo, and push to the remote Git repo, if it's provisioned.

This diagram depicts the relationship between an Azure DevOps Git repo and a Machine Learning project:



To get started using a remote Git repo, complete the steps that are described in the following sections.

NOTE

Currently, Azure Machine Learning supports Git repositories only on Azure DevOps organizations.

Step 1. Create a Machine Learning Experimentation account

Create a Machine Learning Experimentation account and install the Azure Machine Learning Workbench app. For more information, see [Install and create Quickstart](#).

Step 2. Create an Azure DevOps project or use an existing project

In the [Azure portal](#), create a new project:

1. Select +.
2. Search for **Team Project**.
3. Enter the required information:
 - **Name:** A team name.
 - **Version Control:** Select **Git**.
 - **Subscription:** Select a subscription that has a Machine Learning Experimentation account.
 - **Location:** Ideally, choose a region that is close to your Machine Learning Experimentation resources.
4. Select **Create**.

Team Project (preview)

Visual Studio Team Services provides you and your team a set of cloud-powered collaboration tools that work with your existing IDE or editor, so your team can work effectively on software projects of all shapes and sizes. Team Services includes everything from hosted code repos, project tracking tools and agile portfolio management tools, to continuous integration, test case management and cloud based load testing. It scales to meet any team size and provides the capabilities to be the core of your ALM and DevOps practices.

Included with your FREE Team Services account:

- 5 FREE Team Services users
- Unlimited FREE stakeholders
- Unlimited FREE eligible MSDN subscribers
- Unlimited private code repos with Git of Team Foundation Version Control
- FREE 240 minutes/month of build
- FREE 20K virtual user minutes/month of load testing

[Twitter](#) [Facebook](#) [LinkedIn](#) [YouTube](#) [Google+](#) [Email](#)

TeamProject1 **MicroWebsite4_CD_20140329.1**

Summary

Code

Associated items

Build results

Properties

Commits

Deployments

Work items

Create

New Team Project PREVIEW

Add code repositories, plan and track work items, and automatically build and deploy.

Name

* Account [AzureMLUser](#)

Account Owner

* Version Control [Git](#)

* Process Template [Scrum](#)

* Resource Group [VS-AzureMLUser-Group](#)

* Subscription [AML V1 Personal 1](#)

* Location [Central US](#)

Pin to dashboard

Create

Ensure that you sign in by using the same Azure Active Directory (Azure AD) account that you use to access Machine Learning Workbench. Otherwise, the system cannot access Machine Learning Workbench by using your Azure AD credentials. An exception is if you use the command line to create the Machine Learning project, and supply a personal access token to access the Git repo. We discuss this in more detail later in the article.

To go directly to the project that you created, use the URL <https://<project name>.visualstudio.com>.

Step 3. Set up a Machine Learning project and Git repo

To set up a Machine Learning project, you have two options:

- Create a Machine Learning project that has a remote Git repo
- Associate an existing Machine Learning project with an Azure DevOps Git repo

Create a Machine Learning project that has a remote Git repo

Open Machine Learning Workbench and create a new project. In the **Git repo** box, enter the Azure DevOps Git repo URL from Step 2. It usually looks like this: https://<Azure DevOps organization name>.visualstudio.com/_git/<project name>

Create New Project

X

Project name

Project directory

[Browse...](#)

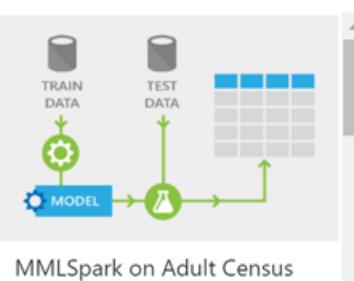
Project description (optional)

Visualstudio.com GIT Repository URL^①

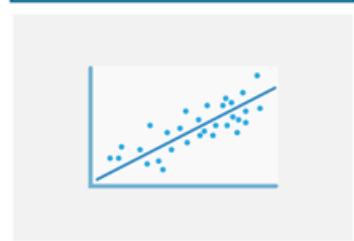
Workspace

[▼](#)[See all in Gallery](#)

Blank Project



MMLSpark on Adult Census

[Create](#)[Cancel](#)

You can also create the project by using the Azure command-line tool (Azure CLI). You have the option of entering a personal access token. Machine Learning can use this token to access the Git repo instead of using your Azure AD credentials:

```
# Create a new project that has a Git repo by using a personal access token.  
$ az ml project create -a <Experimentation account name> -n <project name> -g <resource group name> -w <workspace name> -r <Git repo URL> --vststoken <Azure DevOps personal access token>
```

IMPORTANT

If you choose the blank project template, the Git repo you choose to use might already have a master branch. Machine Learning simply clones the master branch locally. It adds the aml_config folder and other project metadata files to the local project folder.

If you choose any other project template, your Git repo *cannot* already have a master branch. If it does, you see an error. The alternative is to use the `az ml project create` command to create the project, with a `--force` switch. This deletes the files in the original master branch and replaces them with the new files in the template that you choose.

A new Machine Learning project is created, with remote Git repo integration enabled. The project folder is always Git-initialized as a local Git repo. The Git remote is set to the remote Azure DevOps Git repo, so you can push commits to the remote Git repo.

Associate an existing Machine Learning project with an Azure DevOps Git repo

You can create a Machine Learning project without an Azure DevOps Git repo, and rely on the local Git repo for run history snapshots. Later, you can associate an Azure DevOps Git repo with this existing Machine Learning project by using the following command:

```
# Ensure that you are in the project path so Azure CLI has the context of your current project.  
$ az ml project update --repo https://<Azure DevOps organization name>.visualstudio.com/_git/<project name>
```

NOTE

You can perform the update-repo operation only on a Machine Learning project that doesn't have a Git repo associated with it. Also, after you associate a Git repo with a Machine Learning, you can't remove it.

Step 4. Capture a project snapshot in the Git repo

Execute a few script runs in the project, and make some changes in between the runs. You can do this either in the desktop app, or from Azure CLI by using the `az ml experiment submit` command. For more information, see the [Classifying Iris tutorial](#). For each run, if any change is made in any file in the project folder, a snapshot of the entire project folder is committed and pushed to the remote Git repo under a branch named `AzureMLHistory/<project GUID>`. To view the branches and commits, including the `AzureMLHistory/<project GUID>` branch, go to the Azure DevOps Git repo URL.

NOTE

The snapshot is committed only before a script execution. Currently, a data prep execution or a Notebook cell execution doesn't trigger the snapshot.

The screenshot shows the Azure ML History view in Visual Studio Code. The left sidebar lists files like .azureml, aml_config, images, .gitignore, iris.csv, iris.dprep, iris.dprep.user, iris.dsouce, iris.dsouce.user, iris.ipynb, iris_plot_lib.py, iris_pyspark.py, iris_score.py, iris_sklearn.py, iris_sklearn_shared_folder.py, readme.md, and run.py. The main area displays a commit history with three entries:

Commit	Message	Author	Authored Date	Pull Request	Build
b859f697		ecaterina@bellow...	just now		
d58e2115		ecaterina@bellow...	a minute ago		
a1e17440	Initial commit	ecaterina@bellow...	4 minutes ago		

IMPORTANT

It's best if you don't operate in the history branch by using Git commands. It might interfere with the run history. Instead, use the master branch or create other branches for your own Git operations.

Step 5. Restore a previous project snapshot

To restore the entire project folder to the state of a previous run history snapshot, in Machine Learning Workbench:

1. In the activity bar (hourglass icon), select **Runs**.
2. In the **Run List** view, select the run that you want to restore.
3. In the **Run Detail** view, select **Restore**.

The screenshot shows the Azure Machine Learning Workbench (Preview) interface. The left sidebar shows a navigation tree with 'Demo' and 'RiskAnalysis'. The main area has tabs for 'Project Dashboard', 'iris_sklearn.py', and 'iris_sklearn.py'. Below these are buttons for 'Run List', 'Compare', and 'Restore'. The 'Run Properties' section displays the following details for a completed run:

Status	Completed
Start Time	Sep 14, 2017, 11:10:06 PM
Duration	9s
Target	local
Run Id	RiskAnalysis_1505445006107
Run Number	5
Script Name	iris_sklearn.py
Git Commit Hash	b859f697c356ca5f53df412e114f3b9fb48bb4d
Regularization Rate	0.01
Accuracy	0.6792452830188679

The 'Outputs' section shows two items: 'model.pkl' and 'sdk_debug.txt'.

Optionally, you can use the following commands in the Azure CLI window in Machine Learning Workbench:

```
# Discover the run I want to restore a snapshot from.  
$ az ml history list -o table  
  
# Restore the snapshot from a specific run.  
$ az ml project restore --run-id <run ID>
```

Be cautious when you run this command. Executing this command overwrites the entire project folder with the snapshot that was taken when that specific run was kicked off. Your project stays in the current branch. This means that you **lose all changes** in your current project folder.

You might want to use Git to commit your changes to the current branch before you perform this operation.

Step 6. Use the master branch

One way to avoid accidentally losing your current project state is to commit the project to the master branch of the Git repo (or to any branch that you created yourself). You can use Git from the command line or from your favorite Git client tool to operate on the master branch. For example:

```
# Check status to make sure you are on the master branch (or branch of your choice).  
$ git status  
  
# Stage all changes.  
$ git add -A  
  
# Commit all changes locally on the master branch.  
$ git commit -m 'these are my updates so far'  
  
# Push changes to the remote Azure DevOps Git repo master branch.  
$ git push origin master
```

Now, you can safely restore the project to an earlier snapshot by completing Step 5. You can always come back to the commit you just made on the master branch.

Authentication

If you rely only on the run history functions in Machine Learning to take project snapshots and restore them, you don't need to worry about Git repo authentication. Authentication is handled by the Machine Learning Experimentation service layer.

However, if you use your own Git tools to manage version control, you need to handle authentication against the remote Git repo in Azure DevOps. In Machine Learning, the remote Git repo is added to the local repo as a Git remote by using the HTTPS protocol. This means that when you issue Git commands (such as push or pull) to the remote, you need to provide your user name and password, or a personal access token. To create a personal access token in an Azure DevOps Git repo, follow the instructions in [Use a personal access token to authenticate](#).

Next steps

- Learn how to [use the Team Data Science Process to organize your project structure](#).

Structure projects with the Team Data Science Process template

11/7/2018 • 5 minutes to read • [Edit Online](#)

This document provides instructions on how to create data science projects in an earlier version of Azure Machine Learning using Team Data Science Process (TDSP) templates. These templates help to structure projects for collaboration and reproducibility.

NOTE

This article is deprecated. Support for earlier versions of this service will end incrementally. [View the support timeline.](#)
Start using the latest version with this [quickstart](#).

What is the Team Data Science Process?

The TDSP is an agile, iterative, data science process for executing and delivering advanced analytics solutions. It's designed to improve the collaboration and efficiency of data science teams in enterprise organizations. It supports these objectives with four key components:

- A standard [data science lifecycle](#) definition.
- A standardized project structure, [project documentation, and reporting templates](#).
- An infrastructure and resources for project execution, such as, respectively, a compute and storage infrastructure and code repositories.
- [Tools and utilities](#) for data science project tasks, such as:
 - Collaborative version control
 - Code review
 - Data exploration and modeling
 - Work planning

For a more complete discussion of the TDSP, see the [Team Data Science Process overview](#).

Why should you use the TDSP structure and templates?

Standardization of the structure, lifecycle, and documentation of data science projects is key to facilitating effective collaboration on data science teams. Create machine learning projects with the TDSP template to provide such a framework for coordinated teamwork.

We previously released a [GitHub repository for the TDSP project structure and templates](#) to help achieve these objectives. But it was not possible, until now, to instantiate the TDSP structure and templates within a data science tool. It's now possible to create a machine learning project that instantiates the TDSP structure and documentation templates.

Things to note before creating a new project

Review the following items *before* you create a new project:

- Review the TDSP Machine Learning [template](#).
- The contents (other than what is already present in the "docs" folder) are required to be less than 25 MB in size.
See the note that follows this list.

- The sample_data folder is only for small data files (less than 5 MB) with which you can test your code or start early development.
- Storing files, such as Word and PowerPoint files, can increase the size of the "docs" folder substantially. We advise that you to find a collaborative Wiki, [SharePoint](#), or other collaborative resource to store such files.
- To learn how to handle large files and outputs in Machine Learning, read [Persisting changes and dealing with large files](#).

NOTE

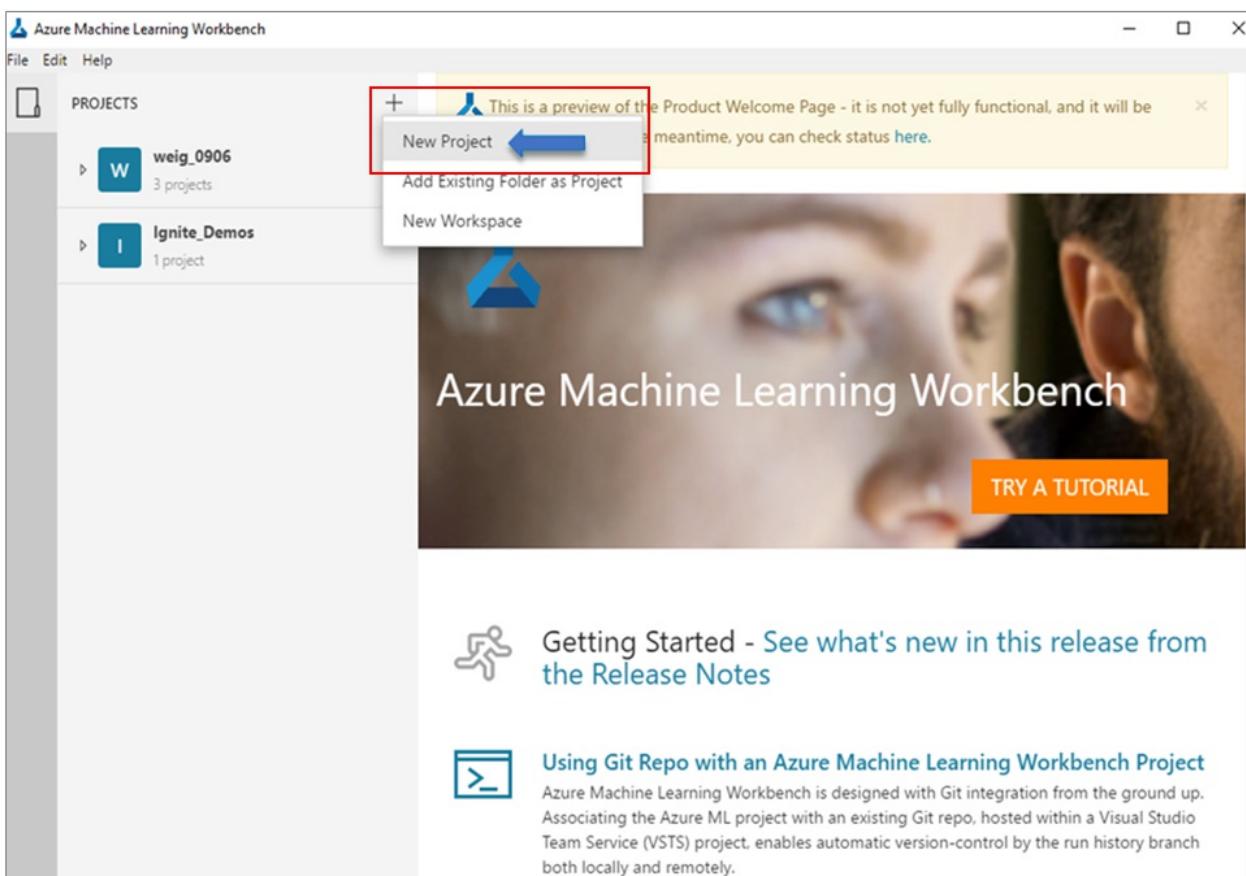
All documentation-related content (text, markdowns, images, and other document files) that is *not* used during project execution, other than the readme.md file, must reside in the folder named "docs" (all lowercase). The "docs" folder is a special folder ignored by Machine Learning execution so that the contents in this folder don't get copied to compute targets unnecessarily. Objects in this folder also don't count toward the 25 MB cap for the project size. The "docs" folder, for example, is the place to store large image files needed in your documentation. These files are still tracked by Git through the run history.

Instantiate the TDSP structure and templates from the Machine Learning template gallery

To create a new project with the TDSP structure and documentation templates, complete the following procedures.

Create a new project

To create a new project, open Azure Machine Learning. Under **Projects** on the top-left pane, select the plus sign (+), and then select **New Project**.



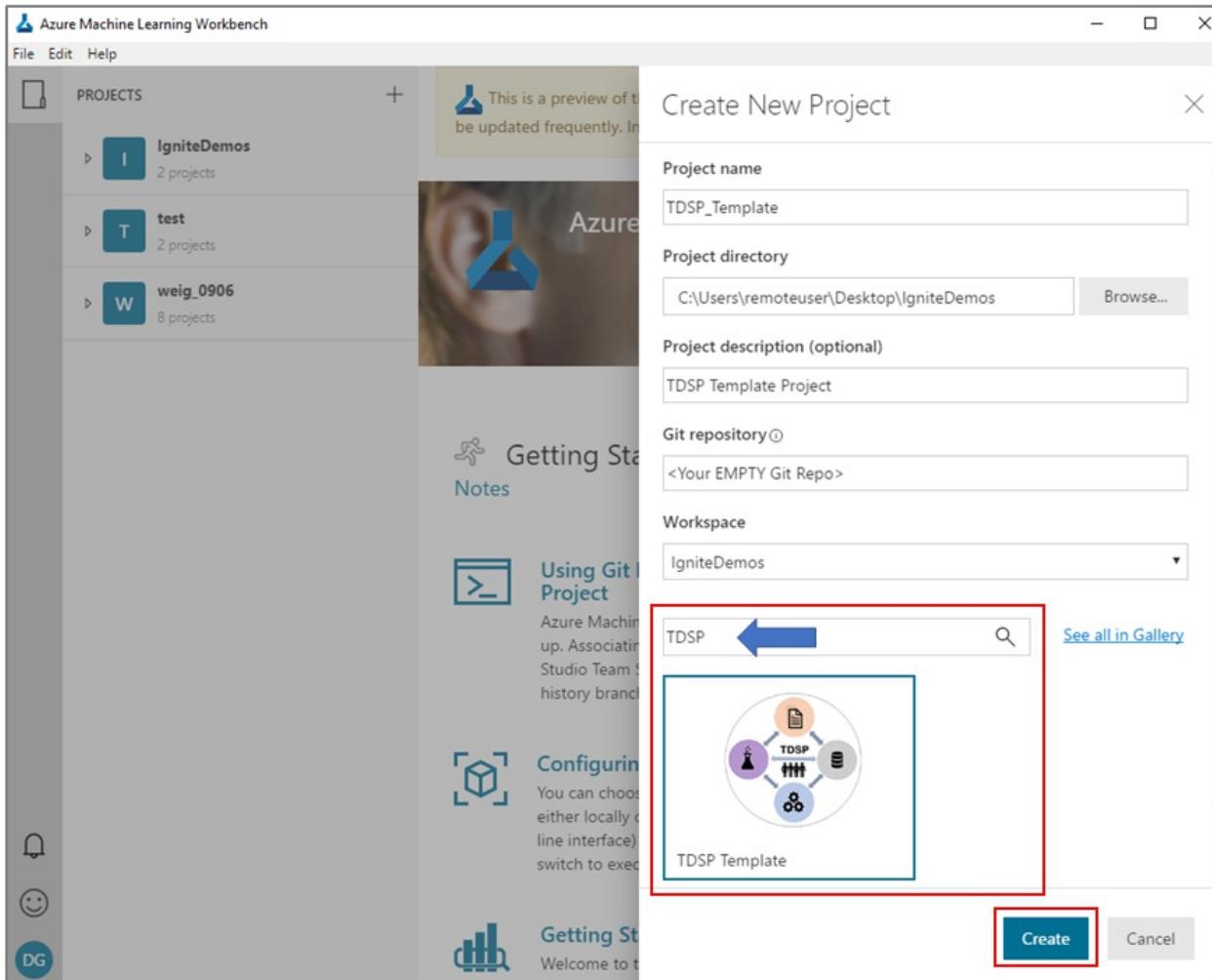
Create a new TDSP-structured project

1. Specify the parameters and information in the relevant box or list:
 - Project name

- Project directory
- Project description
- An empty Git repository path
- Workspace name

2. Then in the **Search** box, enter **TDSP**.

3. When the **Structure a project with TDSP** option appears, select that template.
4. Select the **Create** button to create your new project with a TDSP structure. If you provide an empty Git repository when you create the project (in the appropriate text box), then that repository will populate with the project structure and contents after creation of the project.



Examine the TDSP project structure

After your new project has been created, you can examine its structure (see the left panel in the following figure). It contains all the aspects of standardized documentation for business understanding. These items include the stages of the TDSP lifecycle, data location, definitions, and the architecture of this documentation template.

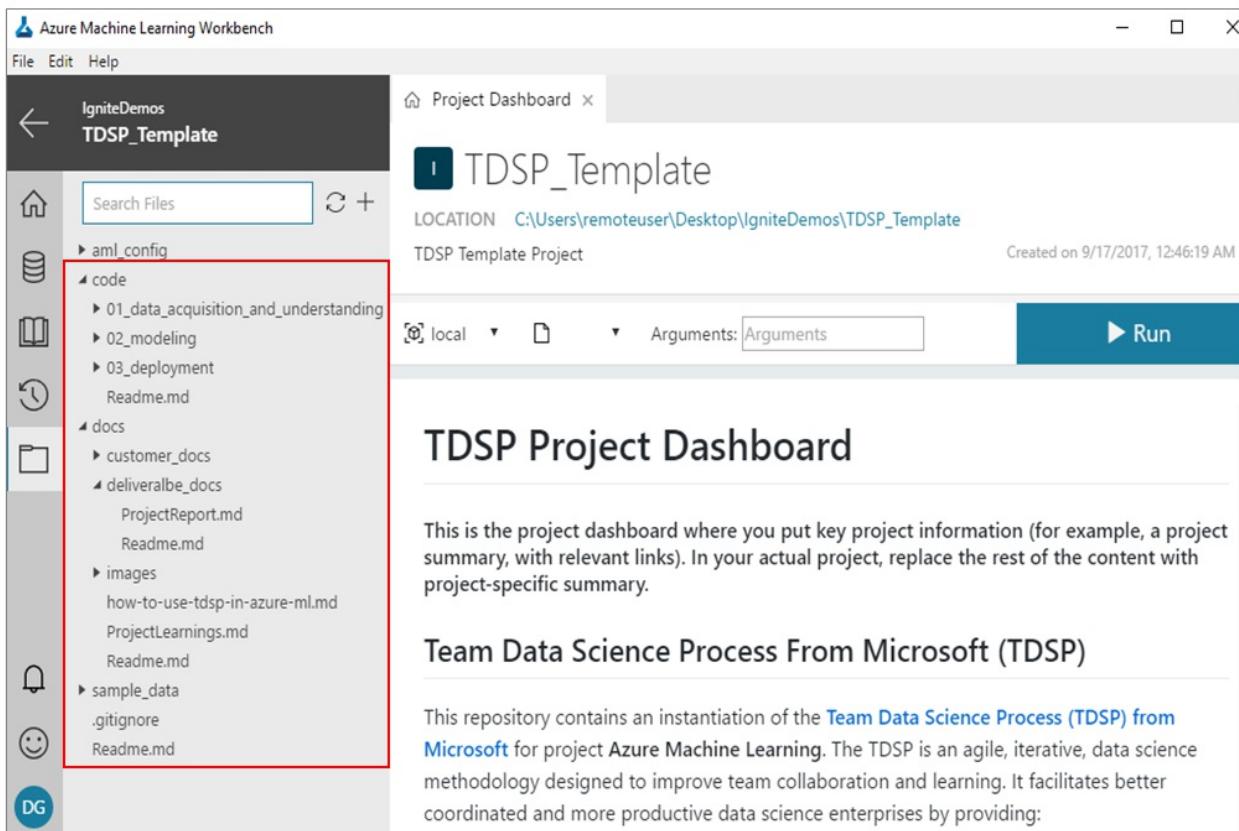
The structure shown is derived from the TDSP structure that is published in [TDSP project structure, documents, and artifact templates](#), with some modifications. For example, several of the document templates are merged into one markdown, namely, [ProjectReport](#).

Project folder structure

The TDSP project template contains the following top-level folders:

- **code**: Contains code.
- **docs**: Contains necessary documentation about the project (for example, markdown files and related media).

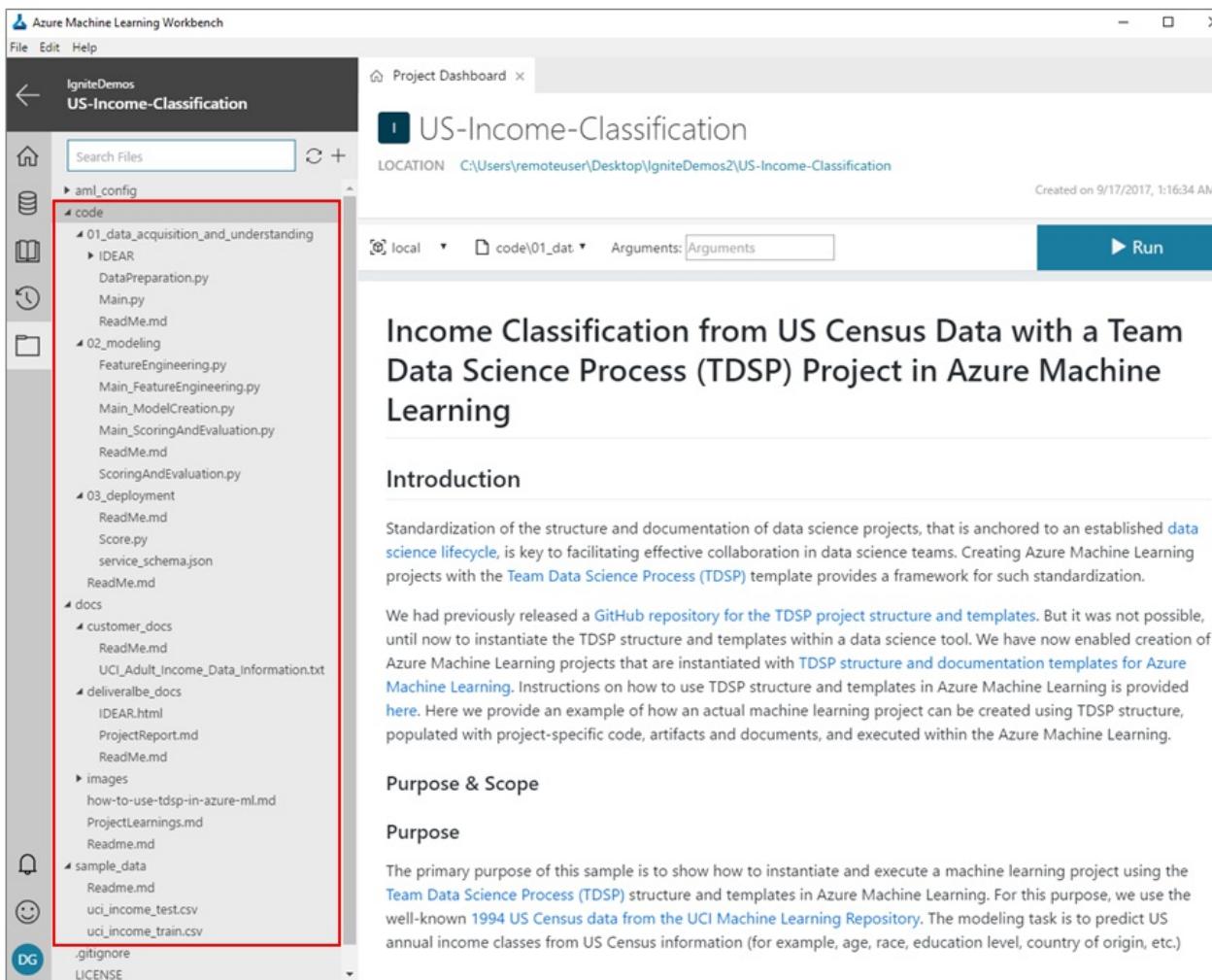
- **sample_data**: Contains **SAMPLE (small)** data that you can use for early development or testing. Typically, these sets are not larger than several (5) MB. This folder is not for full or large data sets.



Use the TDSP structure and templates

You need to add project-specific information to the structure and templates. You're expected to populate these with the code and the information necessary to execute and deliver your project. The [ProjectReport](#) file is a template that you need to modify with information relevant to your project. It comes with a set of questions that help you fill out the information for each of the four stages of the [TDSP lifecycle](#).

An example of what a project structure looks like during execution or after completion is shown in the left panel in the following figure. This project is from the [Team Data Science Process sample project: Classify incomes from US census data in Azure Machine Learning](#) sample project.



Document your project

Refer to the [TDSP documentation templates](#) for information on how to document your project. In the current Machine Learning TDSP documentation template, we recommend that you include all the information in the [ProjectReport](#) file. This template should be filled out with information that is specific to your project.

We also provide a [ProjectLearnings](#) template. You can use this template to include any information that is not included in the primary project document, but that is still useful to document.

Example project report

You can get an [example project report](#). This project report for the [US income classification sample project](#) shows how to instantiate and use the TDSP template for a data science project.

Next steps

To facilitate your understanding on how to use the TDSP structure and templates in Machine Learning projects, we provide several completed project examples in the documentation for Machine Learning:

- For a sample that shows how to create a TDSP project in Machine Learning, see [Team Data Science Process sample project: Classify incomes from US Census data in Azure Machine Learning](#).
- For a sample that uses deep learning in natural language processing (NLP) in a TDSP-instantiated project in Machine Learning, see [Bio-medical entity recognition using natural language processing with deep learning](#).

Roaming and collaboration in Azure Machine Learning Workbench

12/1/2018 • 9 minutes to read • [Edit Online](#)

NOTE

This article is deprecated. Support for earlier versions of this service will end incrementally. [View the support timeline](#). Start using the latest version with this [quickstart](#).

This article describes how you can use Azure Machine Learning Workbench to set up projects for roaming between computers and collaborate with team members.

When you create an Azure Machine Learning project that has a remote Git repository (repo) link, the project metadata and snapshots are stored in the cloud. You can use the cloud link to access the project from a different computer (roaming). You can also collaborate with team members by giving them access to the project.

Prerequisites

1. Install the Machine Learning Workbench app. Ensure that you have access to an Azure Machine Learning Experimentation account. For more information, see the [Installation guide](#).
2. Access [Azure DevOps](#) and then create a repo to link your project to. For more information, see [Using a Git repo with a Machine Learning Workbench project](#).

Create a new Machine Learning project

Open Machine Learning Workbench, and then create a new project (for example, a project named `iris`). In the **Visualstudio.com GIT Repository URL** box, enter a valid URL for an Azure DevOps Git repo.

IMPORTANT

If you choose the blank project template, the Git repo you choose to use might already have a master branch. Machine Learning simply clones the master branch locally. It adds the `aml_config` folder and other project metadata files to the local project folder.

If you choose any other project template, your Git repo *cannot* already have a master branch. If it does, you see an error. The alternative is to use the `az ml project create` command to create the project, with a `--force` switch. This deletes the files in the original master branch and replaces them with the new files in the template that you choose.

After the project is created, submit a few runs on any scripts that are in the project. This action commits the project state to the remote Git repo's run history branch.

NOTE

Only script runs trigger commits to the run history branch. Data prep execution and Notebook runs don't trigger project snapshots in the run history branch.

If you have set up Git authentication, you can also operate in the master branch. Or, you can create a new branch.

Example:

```
# Check current repo status.  
$ git status  
  
# Stage all changes in the current repo.  
$ git add -A  
  
# Commit changes.  
$ git commit -m "my commit fixes this weird bug!"  
  
# Push to the remote repo.  
$ git push origin master
```

Roaming

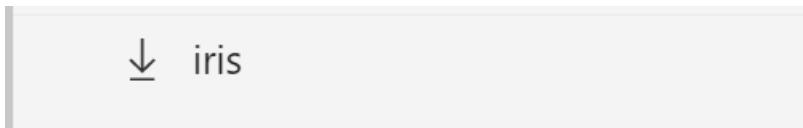
Open Machine Learning Workbench on a second computer

After the Azure DevOps Git repo is linked with your project, you can access the iris project from any computer that has Machine Learning Workbench installed.

To access the iris project on another computer, you must sign in to the app by using the same credentials that you used to create the project. You also need to be in the same Machine Learning Experimentation account and workspace. The iris project is alphabetically listed with other projects in the workspace.

Download the project on a second computer

When the workspace is open on the second computer, the icon adjacent to the iris project is different from the typical folder icon. The download icon indicates that the contents of the project are in the cloud, and that the project is ready to be downloaded to the current computer.



Select the iris project to begin a download. When the download is finished, the project is ready to be accessed on the second computer.

On Windows, the project is located at C:\Users\<username>\Documents\AzureML.

On macOS, the project is located at /home/<username>/Documents/AzureML.

In a future release, we plan to enhance functionality so that you can select a destination folder.

NOTE

If you have a folder in the Machine Learning directory that has the exact same name as the project, the download fails. To work around this issue, temporarily rename the existing folder.

Work on the downloaded project

The newly downloaded project reflects the project state at the last run in the project. A snapshot of the project state is automatically committed to the run history branch in the Azure DevOps Git repo every time you submit a run. The snapshot that is associated with the latest run is used to instantiate the project on the second computer.

Collaboration

You can collaborate with team members on projects that are linked to an Azure DevOps Git repo. You can assign permissions to users for the Machine Learning Experimentation account, workspace, and project. Currently, you

can perform Azure Resource Manager commands by using Azure CLI. You can also use the [Azure portal](#). For more information, see [Use the Azure portal to add users](#).

Use the command line to add users

As an example, Alice is the Owner of the iris project. Alice wants to share access to the project with Bob.

Alice selects the **File** menu, and then selects the **Command Prompt** menu item. The Command Prompt window opens with the iris project. Alice can then decide what level of access she wants to give to Bob. She grants permissions by executing the following commands:

```
# Find the Resource Manager ID of the Experimentation account.  
az ml account experimentation show --query "id"  
  
# Add Bob to the Experimentation account as a Contributor.  
# Bob now has read/write access to all workspaces and projects under the account by inheritance.  
az role assignment create --assignee bob@contoso.com --role Contributor --scope <Experimentation account  
Resource Manager ID>  
  
# Find the Resource Manager ID of the workspace.  
az ml workspace show --query "id"  
  
# Add Bob to the workspace as an Owner.  
# Bob now has read/write access to all projects under the workspace by inheritance. Bob can invite or remove  
other users.  
az role assignment create --assignee bob@contoso.com --role Owner --scope <workspace Resource Manager ID>
```

After role assignment, either directly or by inheritance, Bob can see the project in the Machine Learning Workbench project list. Bob might need to restart the application to see the project. Bob can then download the project as described in [Roaming](#), and begin to collaborate with Alice.

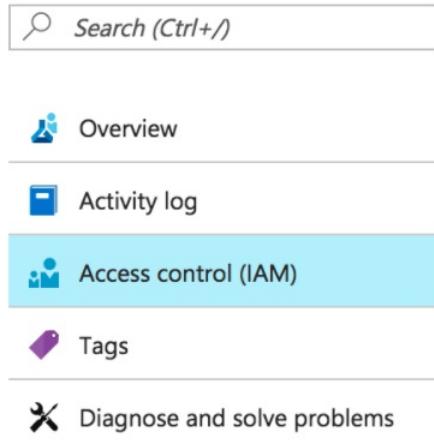
The run history for all users that collaborate on a project is committed to the same remote Git repo. When Alice executes a run, Bob can see the run in the run history section of the project in the Machine Learning Workbench app. Bob can also restore the project to the state of any run, including runs that Alice started.

By sharing a remote Git repo for the project, Alice and Bob can also collaborate in the master branch. If needed, they can also create personal branches and use Git pull requests and merges to collaborate.

Use the Azure portal to add users

Machine Learning Experimentation accounts, workspaces, and projects are Azure Resource Manager resources. To assign roles, you can use the **Access control (IAM)** link in the [Azure portal](#).

Find the resource that you want to add users to by using the **All Resources** view. Select the **Access control (IAM)** link, and then select **Add role assignment**.



Sample collaboration workflow

To illustrate the collaboration workflow, let's walk through an example. Contoso employees Alice and Bob want to collaborate on a data science project by using Machine Learning Workbench. Their identities belong to the same Contoso Azure Active Directory (Azure AD) tenant. Here are the steps Alice and Bob take:

1. Alice creates an empty Git repo in an Azure DevOps project. The Azure DevOps project should be in an Azure subscription that is created under the Contoso Azure AD tenant.
2. Alice creates a Machine Learning Experimentation account, a workspace, and a Machine Learning Workbench project on her computer. When she creates the project, she enters the Azure DevOps Git repo URL.
3. Alice starts to work on the project. She creates some scripts and executes a few runs. For each run, a snapshot of the entire project folder is automatically pushed as a commit to a run history branch of the Azure DevOps Git repo that Machine Learning Workbench creates.
4. Alice is happy with the work in progress. She wants to commit her changes in the local master branch and then push them to Azure DevOps Git repo master branch. With the project open, in Machine Learning Workbench, she opens the Command Prompt window, and then enters these commands:

```
# Verify that the Git remote is pointing to the Azure DevOps Git repo.  
$ git remote -v  
  
# Verify that the current branch is master.  
$ git branch  
  
# Stage all changes.  
$ git add -A  
  
# Commit changes with a comment.  
$ git commit -m "this is a good milestone"  
  
# Push the commit to the master branch of the remote Git repo in Azure DevOps.  
$ git push
```

5. Alice adds Bob to the workspace as a Contributor. She can do this in the Azure portal, or by using the `az role assignment` command, as demonstrated earlier. Alice also grants Bob read/write permissions to the Azure DevOps Git repo.
6. Bob signs in to Machine Learning Workbench on his computer. He can see the workspace that Alice shared with him. He can see the iris project listed under that workspace.
7. Bob selects the project name. The project is downloaded to his computer.
 - The downloaded project files are a copy of the snapshot of the latest run that's recorded in the run history. They are not the last commit on the master branch.
 - The local project folder is set to the master branch, with the unstaged changes.
8. Bob can browse runs that were executed by Alice. He can restore snapshots of any earlier runs.
9. Bob wants to get the latest changes that Alice pushed, and then start working in a different branch. In Machine Learning Workbench, Bob opens a Command Prompt window and executes the following commands:

```
# Verify that the Git remote is pointing to the Azure DevOps Git repo.  
$ git remote -v  
  
# Verify that the current branch is master.  
$ git branch  
  
# Get the latest commit in the Azure DevOps Git master branch and overwrite current files.  
$ git pull --force  
  
# Create a new local branch named "bob" so that Bob's work is done in the "bob" branch  
$ git checkout -b bob
```

10. Bob modifies the project and submits new runs. The changes are made on the bob branch. Bob's runs also become visible to Alice.
11. Bob is ready to push his changes to the remote Git repo. To avoid conflict with the master branch, where Alice is working, Bob pushes his work to a new remote branch, which is also named bob.

```
# Verify that the current branch is "bob," and that it has unstaged changes.  
$ git status  
  
# Stage all changes.  
$ git add -A  
  
# Commit the changes with a comment.  
$ git commit -m "I found a cool new trick."  
  
# Create a new branch on the remote Azure DevOps Git repo, and then push the changes.  
$ git push origin bob
```

12. To tell Alice about the cool new trick in his code, Bob creates a pull request on the remote Git repo from the bob branch to the master branch. Alice can then merge the pull request into the master branch.

Next steps

- Learn more about [using a Git repo with a Machine Learning Workbench project](#).

Visual Studio Tools for AI

11/20/2018 • 3 minutes to read • [Edit Online](#)

NOTE

This article is deprecated. Support for earlier versions of this service will end incrementally. [View the support timeline](#). Start using the latest version with this [quickstart](#).

Visual Studio Tools for AI is a development extension to build, test, and deploy Deep Learning / AI solutions. It features a seamless integration with Azure Machine Learning, notably a run history view, detailing the performance of previous trainings and custom metrics. It offers a samples explorer view, allowing to browse and bootstrap new project with [Microsoft Cognitive Toolkit \(previously known as CNTK\)](#), [Google TensorFlow](#), and other deep-learning framework. Finally, it provides an explorer for compute targets, which enables you to submit jobs to train models on remote environments like Azure Virtual Machines or Linux servers with GPU. It also provides a facilitated access to [Azure Batch AI \(Preview\)](#).

Getting started

To get started, you first need to download and install [Visual Studio](#). Once you have Visual Studio open, do the following steps:

1. Click "Tools" on the menu bar in Visual Studio and select "Extensions and Updates..."
2. Click on "Online" tab and select "Search Visual Studio Marketplace."
3. Search for "Visual Studio for AI."
4. Click on the **Download** button.
5. After installation, restart Visual Studio.

Once Visual Studio is reloaded, the extension is active. [Learn more about finding extensions](#).

NOTE

Visual Studio Tools for AI needs Visual Studio 2015 or 2017, professional or enterprise edition. It does not support Apple OSX version.

Exploring project samples

Visual Studio Tools for AI comes with a samples explorer. The samples explorer makes it easy to discover sample and try them with only a few clicks. To open the explorer, do as follows:

1. In the menu bar, click on **AI Tools**.
2. Click on "Azure Machine Learning Gallery."

A tab with all the Azure ML Samples opens.

Creating a new project from the sample explorer

You can browse different samples and get more information about them. Let's browse until finding the "Classifying Iris" sample. To create a new project based on this sample do as follows:

1. Click on **install** button on the project sample, this will open a new dialogue.

2. Select a resource group, an account, and a workspace.
3. You can leave project type as General.
4. Enter a project path and a project name, then press enter.
5. A dialogue opens prompting to save a solution, click save.

Once complete, a new project opens in a new instance of Visual Studio.

TIP

You need to be logged-in to access your Azure resource. From the embedded terminal enter "az login" and follow the instruction.

Submitting experiment with the new project

The new project being open in Visual Studio, submit a job to a compute target (local or VM with docker). To submit the job, do as follow:

1. From the solution explorer, right-click on the file you want to submit, and select **Set as Startup File**.
2. Select the project name, right-click and select **Submit Job...**
3. A new dialogue will open, letting you choose the cluster (or compute target) to execute your script.
4. Click on **Submit**

Once the job is submitted, the embedded-terminal displays the progress of the runs.

View list of jobs

Once the job is submitted, you can list the jobs from the run history.

1. In **Server Explorer**, click on **AI Tools**.
2. Then select **Azure Machine Learning**
3. Click on the **Jobs** menu.

The Job explorer list all the submitted experiment for this project.

View job details

With the Job explorer view open, click on the first run in the list. This will load the Job Summary panel, and the Logs and Outputs panel.

Next steps

[How to configure Azure Machine Learning to work with an IDE](#)

Visual Studio Code Tools for AI

11/20/2018 • 3 minutes to read • [Edit Online](#)

NOTE

This article is deprecated. Support for earlier versions of this service will end incrementally. [View the support timeline](#). Start using the latest version with this [quickstart](#).

Visual Studio Code Tools for AI is a development extension to build, test, and deploy Deep Learning / AI solutions. It features a seamless integration with Azure Machine Learning, notably a run history view, detailing the performance of previous trainings and custom metrics. It offers a samples explorer view, allowing to browse and bootstrap new project with [Microsoft Cognitive Toolkit \(previously known as CNTK\)](#), [Google TensorFlow](#), and other deep-learning framework. Finally, it provides an explorer for compute targets, which enables you to submit jobs to train models on remote environments like Azure Virtual Machines or Linux servers with GPU.

Getting started

To get started, you first need to download and install [Visual Studio Code](#). Once you have Visual Studio Code open, do the following steps:

1. Click on the extension icon in the activity bar.
2. Search for "Visual Studio Code Tools for AI".
3. Click on the **Install** button.
4. After installation, click on **Reload** button.

Once Visual Studio Code is reloaded, the extension is active. [Learn more about installing extensions](#).

Exploring project samples

Visual Studio Code Tools for AI comes with a samples explorer. The samples explorer makes it easy to discover sample and try them with only a few clicks. To open the explorer, do as follow:

1. Open the command palette (View > **Command Palette** or **Ctrl+Shift+P**).
2. Enter "AI Sample".
3. You get a recommendation for "AI: Open Azure ML Sample Explorer", select it and press enter.

Alternatively, you can click on the samples explorer icon.

Creating a new project from the sample explorer

You can browse different samples and get more information about them. Let's browse until finding the "Classifying Iris" sample. To create a new project based on this sample do the following:

1. Click install button on the project sample, notice the commands being prompted, walking you through the steps of creating a new project.
2. Pick a name for the project, for example "Iris".
3. Choose a folder path to create your project and press enter.
4. Select an existing workspace and press enter.

The project will then be created.

TIP

You will need to be logged-in to access your Azure resource. From the embedded terminal enter "az login" and follow the instruction.

Submitting experiment with the new project

The new project being open in Visual Studio Code, we submit a job to our different compute target (local and VM with docker). Visual Studio Code Tools for AI provides multiple ways to submit an experiment.

1. Context Menu (right click) - **AI: Submit Job**.
2. From the command palette: "AI: Submit Job".
3. Alternatively, you can run the command directly using Azure CLI, Machine Learning Commands, using the embedded terminal.

Open iris_sklearn.py, right click and select **AI: Submit Job**.

1. Select your platform: "Azure Machine Learning".
2. Select your run-configuration: "Docker-Python."

NOTE

If it is the first time you submit a job, you receive a message "No Machine Learning configuration found, creating...". A JSON file is opened, save it (**Ctrl+S**).

Once the job is submitted, the embedded-terminal displays the progress of the runs.

View list of jobs

Once the jobs are submitted, you can list the jobs from the run history.

1. Open the command palette (View > **Command Palette** or **Ctrl+Shift+P**).
2. Enter "AI List."
3. You get a recommendation for "AI: List Jobs", select and press enter.

The Job List View opens and displays all the runs and some related information.

View job details

With the Job List View still open, click on the first run in the list. To deep dive into the results of a job, click on the top **job ID** to see detailed information.

Next steps

[How to configure Azure Machine Learning to work with an IDE](#)

Use the AI Toolkit for Azure IoT Edge

12/11/2018 • 2 minutes to read • [Edit Online](#)

NOTE

This article is deprecated. Support for earlier versions of this service will end incrementally. [View the support timeline](#). Start using the latest version with this [quickstart](#).

Bring the power of artificial intelligence, machine learning, and advanced analytics to the edge. With the AI Toolkit for Azure IoT Edge, you can deploy AI and machine learning where your data lives. Operationalize your model as a Docker container with a REST API, and then push the model to an edge device with Azure IoT Hub. AI Toolkit for Azure IoT Edge brings intelligence, elastic compute, and the power of the cloud to places with limited, intermittent, or no connectivity.

The AI Toolkit for Azure IoT Edge is a collection of scripts, code, and deployable containers. Examples include predictive maintenance, image classification, and speech processing, as well as custom model deployment through Azure Machine Learning and Azure IoT Hub. Models included with the toolkit may be used as-is, but all source code and data are available for developers to customize based on their needs.

The AI Toolkit for Azure IoT Edge public GitHub repository can be found at aka.ms/AI-toolkit.

Configuring Azure Machine Learning Experimentation Service

9/24/2018 • 12 minutes to read • [Edit Online](#)

NOTE

This article is deprecated. Support for earlier versions of this service will end incrementally. [View the support timeline.](#)
Start using the latest version with this [quickstart](#).

Overview

Azure Machine Learning Experimentation Service enables data scientists to execute their experiments using the Azure Machine Learning execution and run management capabilities. It provides a framework for agile experimentation with fast iterations. Azure Machine Learning Workbench allows you to start with local runs on your machine and also an easy path for scaling up and out to other environments such as remote Data Science VMs with GPU or HDInsight Clusters running Spark.

Experimentation Service is built for providing isolated, reproducible, and consistent runs of your experiments. It helps you manage your compute targets, execution environments, and run configurations. By using the Azure Machine Learning Workbench execution and run management capabilities, you can easily move between different environments.

You can execute a Python or PySpark script in a Workbench project locally or at scale in the cloud.

You can run your scripts on:

- Python (3.5.2) environment on your local computer installed by Workbench
- Conda Python environment inside of a Docker container on local computer
- On a Python environment that you own and manage on a remote Linux Machine
- Conda Python environment inside of a Docker container on a remote Linux machine. For example, an [Ubuntu-based DSVM on Azure](#)
- [HDInsight for Spark](#) on Azure

IMPORTANT

Azure Machine Learning Experimentation Service currently supports Python 3.5.2 and Spark 2.1.11 as Python and Spark runtime versions, respectively.

Key concepts in Experimentation Service

It is important to understand the following concepts in Azure Machine Learning experiment execution. In the subsequent sections, we discuss how to use these concepts in detail.

Compute target

A *compute target* specifies where to execute your program such as your desktop, remote Docker on a VM, or a cluster. A compute target needs to be addressable and accessible by you. Workbench gives you the ability to create compute targets and manage them using the Workbench application and the CLI.

`az ml computetarget attach` command in CLI enables you to create a compute target that you can use in your runs.

Supported compute targets are:

- Local Python (3.5.2) environment on your computer installed by Workbench.
- Local Docker on your computer
- User-managed, Python environment on remote Linux-Ubuntu VMs. For example, an [Ubuntu-based DSVM on Azure](#)
- Remote Docker on Linux-Ubuntu VMs. For example, an [Ubuntu-based DSVM on Azure](#)
- [HDInsight for Spark cluster](#) on Azure

Experimentation Service currently supports Python 3.5.2 and Spark 2.1.11 as Python and Spark runtime versions, respectively.

IMPORTANT

Windows VMs running Docker are **not** supported as remote compute targets.

Execution environment

The *execution environment* defines the run time configuration and the dependencies needed to run the program in Workbench.

You manage the local execution environment using your favorite tools and package managers if you're running on the Workbench default runtime.

Conda is used to manage local Docker and remote Docker executions as well as HDInsight-based executions. For these compute targets, the execution environment configuration is managed through

Conda_dependencies.yml and **Spark_dependencies.yml** files. These files are in the **aml_config** folder inside your project.

Supported runtimes for execution environments are:

- Python 3.5.2
- Spark 2.1.11

Run configuration

In addition to the compute target and execution environment, Azure Machine Learning provides a framework to define and change *run configurations*. Different executions of your experiment may require different configuration as part of iterative experimentation. You may be sweeping different parameter ranges, using different data sources, and tuning spark parameters. Experimentation Service provides a framework for managing run configurations.

Running `az ml computetarget attach` command produces two files in your **aml_config** folder in your project: a `.compute` and a `.runconfig` following this convention: `<your_computetarget_name>.compute` and `<your_computetarget_name>.runconfig`. The `.runconfig` file is automatically created for your convenience when you create a compute target. You can create and manage other run configurations using `az ml runconfigurations` command in CLI. You can also create and edit them on your file system.

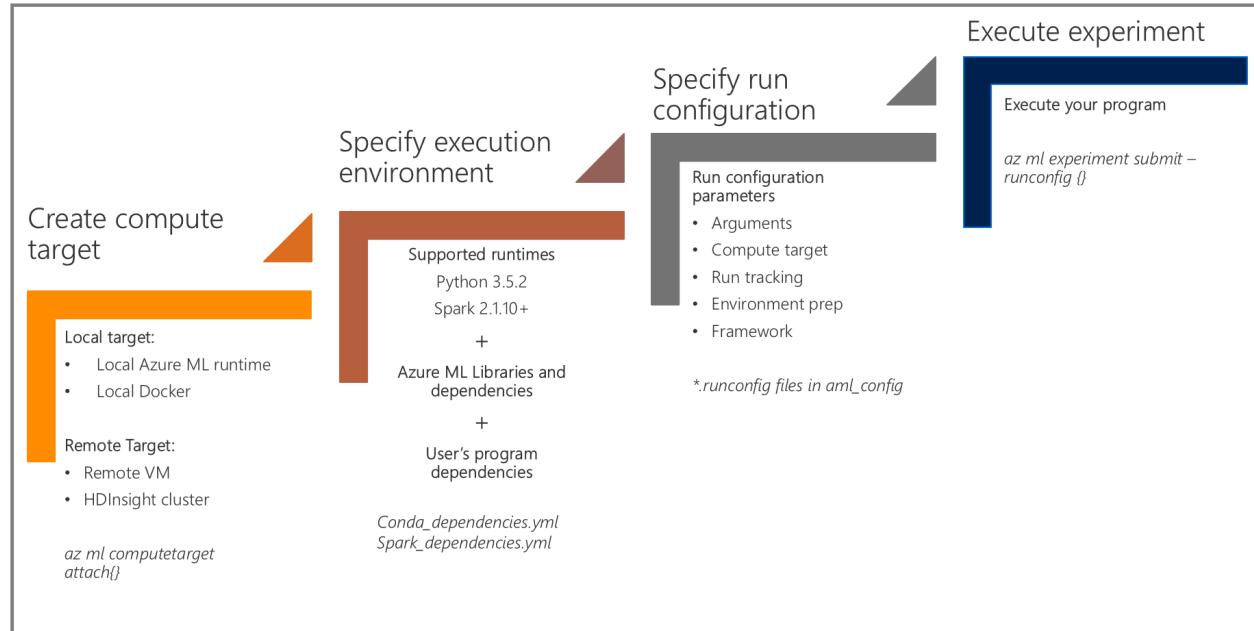
Run configuration in Workbench also enables you to specify environment variables. You can specify environment variables and use them in your code by adding the following section in your `.runconfig` file.

```
EnvironmentVariables:  
    "EXAMPLE_ENV_VAR1": "Example Value1"  
    "EXAMPLE_ENV_VAR2": "Example Value2"
```

These environment variables can be accessed in your code. For example, this python code snippet prints the environment variable named "EXAMPLE_ENV_VAR1"

```
print(os.environ.get("EXAMPLE_ENV_VAR1"))
```

The following figure shows the high-level flow for initial experiment run.



Experiment execution scenarios

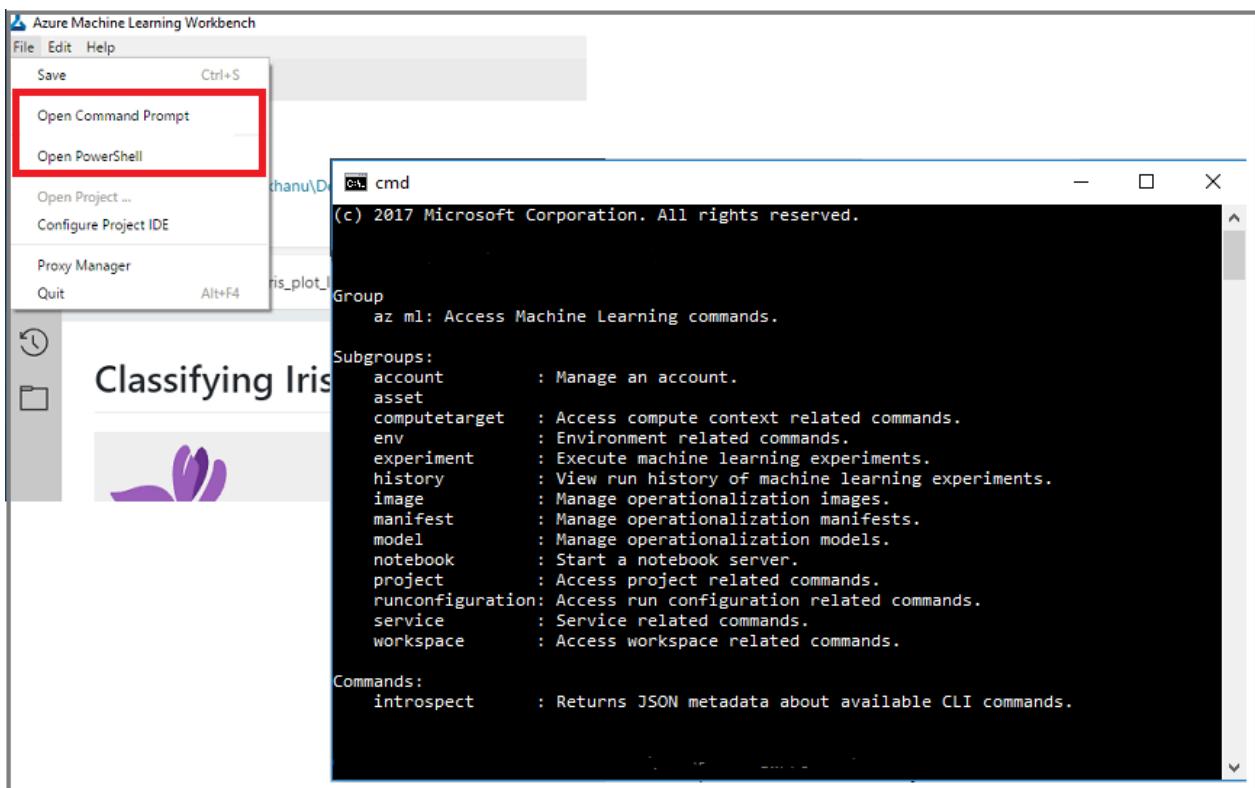
In this section, we dive into execution scenarios and learn about how Azure Machine Learning runs experiments, specifically running an experiment locally, on a remote VM, and on an HDInsight Cluster. This section is a walkthrough starting from creating a compute target to executing your experiments.

NOTE

For the rest of this article, we are using the CLI (Command-line interface) commands to show the concepts and the capabilities. Capabilities described here can also be used from Workbench.

Launching the CLI

An easy way to launch the CLI is opening a project in Workbench and navigating to **File-->Open Command Prompt**.



This command launches a terminal window in which you can enter commands to execute scripts in the current project folder. This terminal window is configured with the Python 3.5.2 environment, which is installed by Workbench.

NOTE

When you execute any `az ml` command from the command window, you need to be authenticated against Azure. CLI uses an independent authentication cache than the desktop app and so logging in to Workbench doesn't mean you are authenticated in your CLI environment. To authenticate, use the following steps. Authentication token is cached locally for a period of time so you only need to repeat these steps when the token expires. When the token expires or if you are seeing authentication errors, execute the following commands:

```
# to authenticate
$ az login

# to list subscriptions
$ az account list -o table

# to set current subscription to a particular subscription ID
$ az account set -s <subscription_id>

# to verify your current Azure subscription
$ az account show
```

NOTE

When you run `az ml` command within a project folder, make sure that the project belongs to an Azure Machine Learning Experimentation account on the *current* Azure subscription. Otherwise you may encounter execution errors.

Running scripts and experiments

With Workbench, you can execute your Python and PySpark scripts on various compute targets using the `az ml experiment submit` command. This command requires a run configuration definition.

Workbench creates a corresponding runconfig file when you create a compute target, but you can create additional run configurations using `az ml runconfiguration create` command. You can also manually edit the run configuration files.

Run configurations show up as part of experiment run experience in Workbench.

NOTE

You can learn more about the run configuration file in the [Experiment Execution Configuration Reference](#) Section.

Running a script locally on Workbench-installed runtime

Workbench enables you to run your scripts directly against the Workbench-installed Python 3.5.2 runtime. This default runtime is installed at Workbench set-up time and includes Azure Machine Learning libraries and dependencies. Run results and artifacts for local executions are still saved in Run History Service in the cloud.

Unlike Docker-based executions, this configuration is *not* managed by Conda. You need to manually provision package dependencies for your local Workbench Python environment.

You can execute the following command to run your script locally in the Workbench-installed Python environment.

```
$az ml experiment submit -c local myscript.py
```

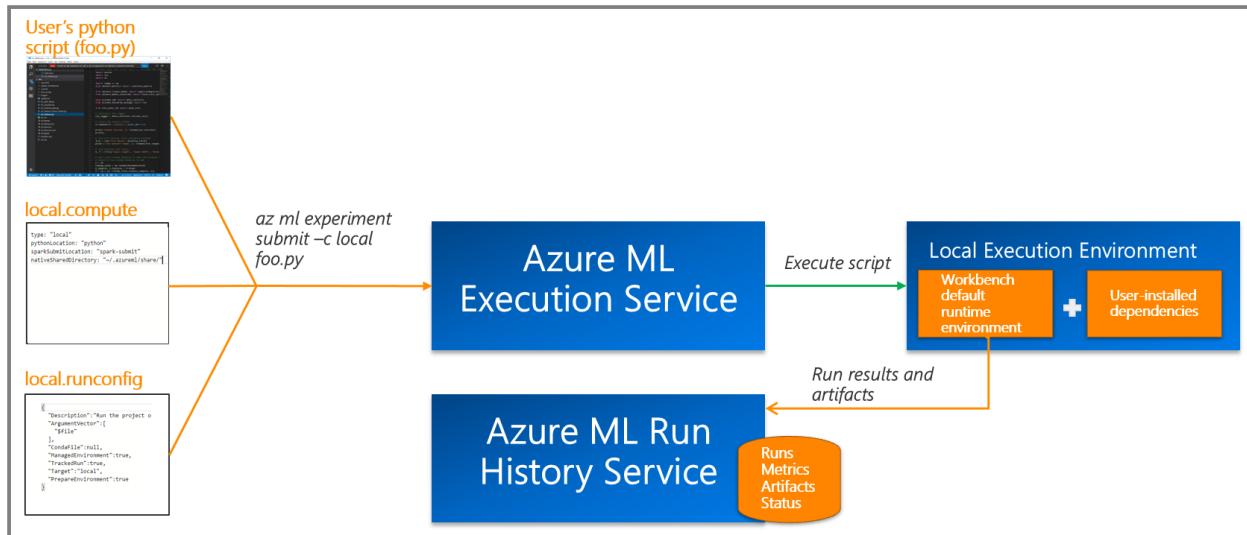
You can find the path to the default Python environment by typing the following command in the Workbench CLI window.

```
$ conda env list
```

NOTE

Running PySpark locally directly against local Spark environments is currently **not** supported. Workbench does support PySpark scripts running on local Docker. Azure Machine Learning base Docker image comes with Spark 2.1.11 pre-installed.

Overview of local execution for a Python script:



Running a script on local Docker

You can also run your projects on a Docker container on your local machine through Experimentation Service. Workbench provides a base Docker image that comes with Azure Machine Learning libraries and as well as Spark 2.1.11 runtime to make local Spark executions easy. Docker needs to be already running on the local machine.

For running your Python or PySpark script on local Docker, you can execute the following commands in CLI.

```
$az ml experiment submit -c docker myscript.py
```

or

```
az ml experiment submit --run-configuration docker myscript.py
```

The execution environment on local Docker is prepared using the Azure Machine Learning base Docker image. Workbench downloads this image when running for the first time and overlays it with packages specified in your `conda_dependencies.yml` file. This operation makes the initial run slower but subsequent runs are considerably faster thanks to Workbench reusing cached layers.

IMPORTANT

You need to run `az ml experiment prepare -c docker` command first to prepare the Docker image for your first run. You can also set the **PrepareEnvironment** parameter to true in your `docker.runconfig` file. This action automatically prepares your environment as part of your run execution.

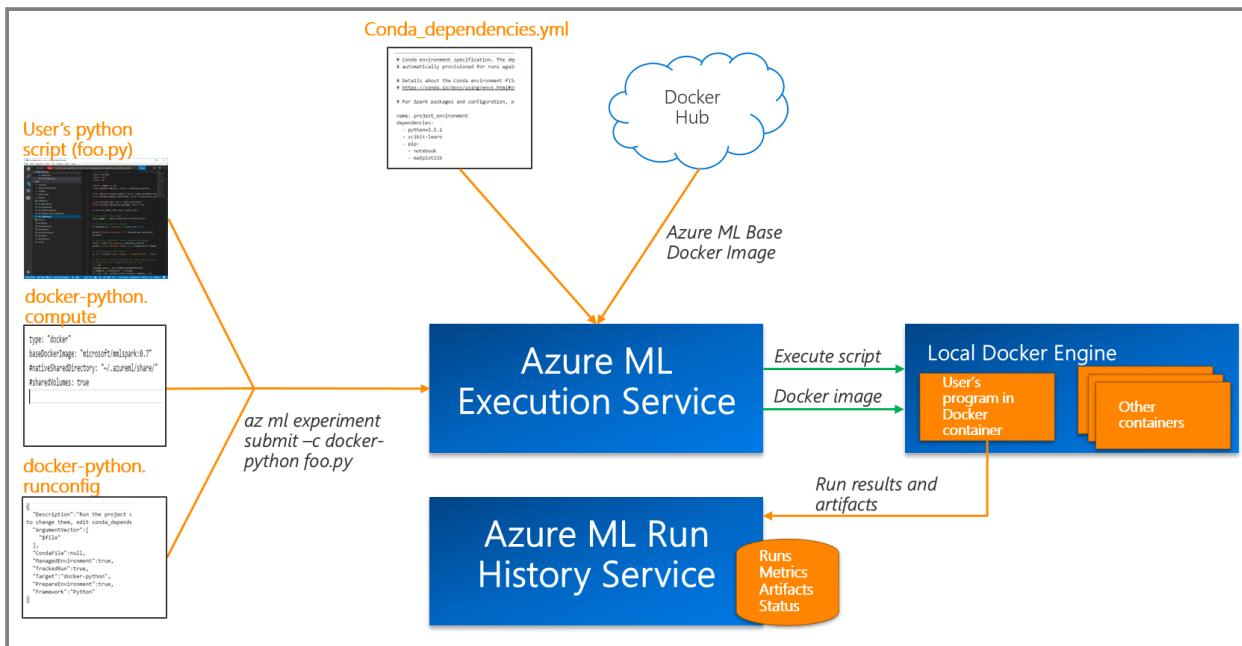
NOTE

If running a PySpark script on Spark, `spark_dependencies.yml` is also used in addition to `conda_dependencies.yml`.

Running your scripts on a Docker image gives you the following benefits:

1. It ensures that your script can be reliably executed in other execution environments. Running on a Docker container helps you discover and avoid any local references that may impact portability.
2. It allows you to quickly test code on runtimes and frameworks that are complex to install and configure, such as Apache Spark, without having to install them yourself.

Overview of local Docker execution for a Python script:



Running a script on a remote Docker

In some cases, resources available on your local machine may not be enough to train the desired model. In this situation, Experimentation Service allows an easy way to run your Python or PySpark scripts on more powerful VMs using remote Docker execution.

Remote VM should satisfy the following requirements:

- Remote VM needs to be running Linux-Ubuntu and should be accessible through SSH.
- Remote VM needs to have Docker running.

IMPORTANT

Windows VMs running Docker is **not** supported as remote compute targets

You can use the following command to create both the compute target definition and run configuration for remote Docker-based executions.

```
az ml computetarget attach remotedocker --name "remotevm" --address "remotevm_IP_address" --username "sshuser" --password "sshpassword"
```

Once you configure the compute target, you can use the following command to run your script.

```
$ az ml experiment submit -c remotevm myscript.py
```

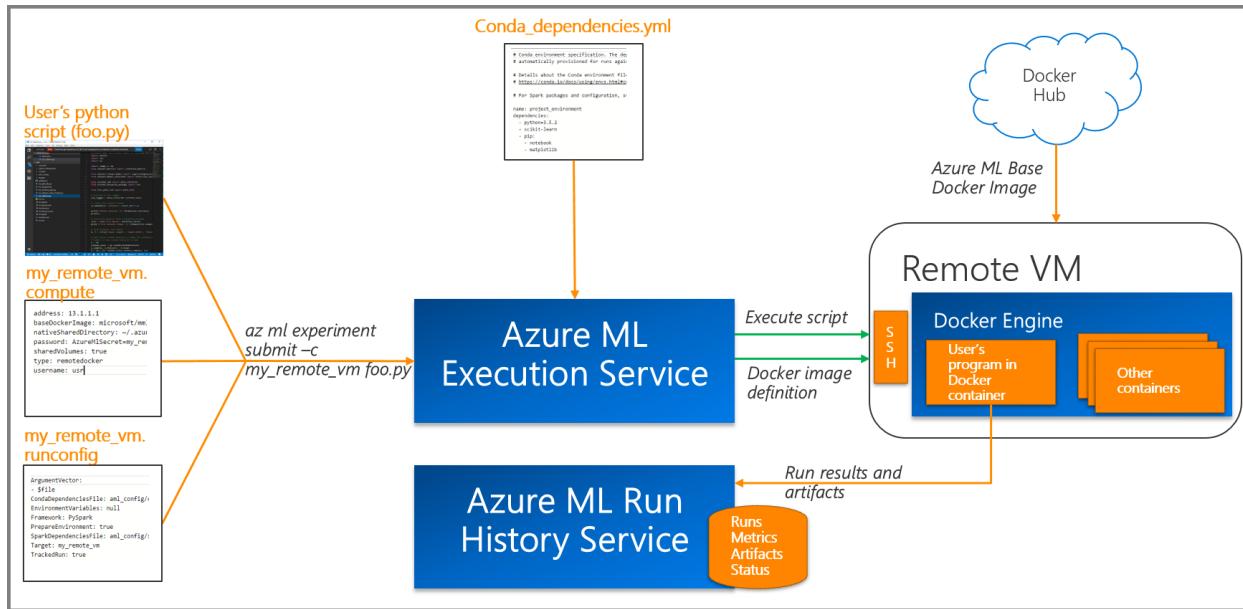
NOTE

Keep in mind that execution environment is configured using the specifications in `conda_dependencies.yml`. `spark_dependencies.yml` is also used if PySpark framework is specified in `.runconfig` file.

The Docker construction process for remote VMs is exactly the same as the process for local Docker runs so you should expect a similar execution experience.

TIP

If you prefer to avoid the latency introduced by building the Docker image for your first run, you can use the following command to prepare the compute target before executing your script. az ml experiment prepare -c remotedocker

Overview of remote vm execution for a Python script:

Running a script on a remote VM targeting user-managed environments

Experimentation service also supports running a script on user's own Python environment inside a remote Ubuntu virtual machine. This allows you to manage your own environment for execution and still use Azure Machine Learning capabilities.

Follow the following steps to run your script on your own environment.

- Prepare your Python environment on a remote Ubuntu VM or a DSVM installing your dependencies.
- Install Azure Machine Learning requirements using the following command.

```
pip install -I --index-url https://azuremldownloads.azureedge.net/python-repository/preview --extra-index-url https://pypi.python.org/simple azureml-requirements
```

TIP

In some cases, you may need to run this command in sudo mode depending on your privileges.

```
sudo pip install -I --index-url https://azuremldownloads.azureedge.net/python-repository/preview --extra-index-url https://pypi.python.org/simple azureml-requirements
```

- Use the following command to create both the compute target definition and run configuration for user-managed runs on remote VM executions.

```
az ml computetarget attach remote --name "remotenvm" --address "remotenvm_IP_address" --username "sshuser" --password "sshpassword"
```

NOTE

This will set "userManagedEnvironment" parameter in your .compute configuration file to true.

- Set location of your Python runtime executable in your .compute file. You should refer to the full path of your python executable.

```
pythonLocation: python3
```

Once you configure the compute target, you can use the following command to run your script.

```
$ az ml experiment submit -c remotevm myscript.py
```

NOTE

When you are running on a DSVM, you should use the following commands

If you would like to run directly on DSVM's global python environment, run this command.

```
sudo /anaconda/envs/py35/bin/pip install <package>
```

Running a script on an HDInsight cluster

HDInsight is a popular platform for big-data analytics supporting Apache Spark. Workbench enables experimentation on big data using HDInsight Spark clusters.

NOTE

The HDInsight cluster must use Azure Blob as the primary storage. Using Azure Data Lake storage is not supported yet.

You can create a compute target and run configuration for an HDInsight Spark cluster using the following command:

```
$ az ml computetarget attach cluster --name "myhdi" --address "<FQDN or IP address>" --username "sshuser" --password "sshpssword"
```

NOTE

If you use FQDN instead of an IP address and your HDI Spark cluster is named *foo*, the SSH endpoint is on the driver node named *foo-ssh.azurehdinsight.net*. Don't forget the **-ssh** postfix in the server name when using FQDN for **--address** parameter.

Once you have the compute context, you can run the following command to execute your PySpark script.

```
$ az ml experiment submit -c myhdi myscript.py
```

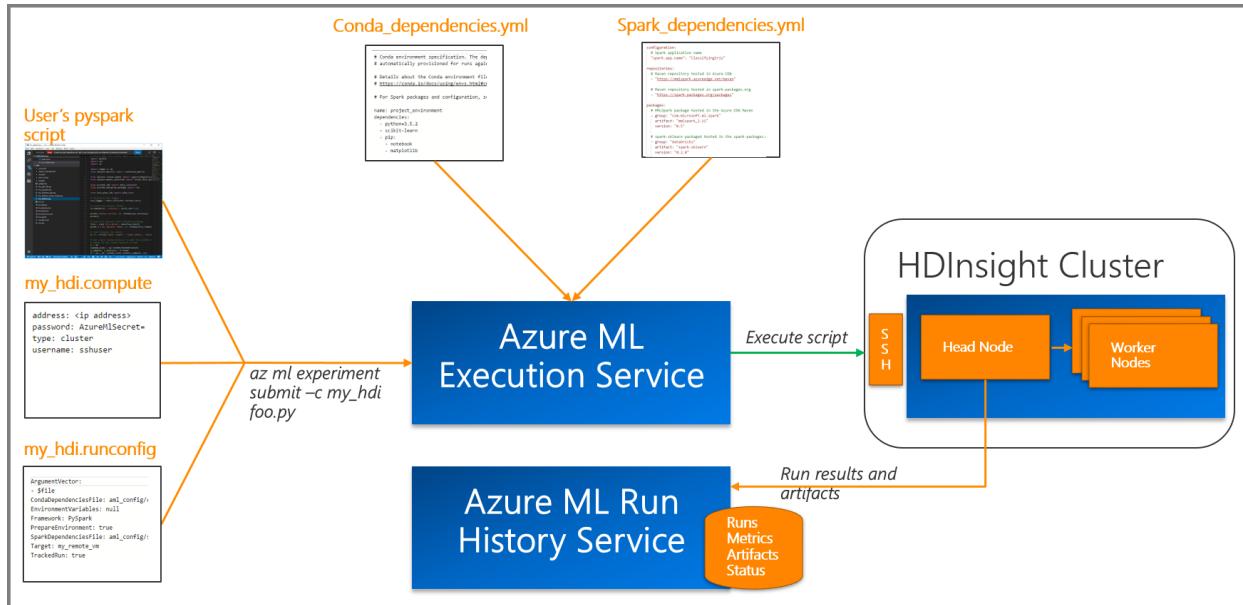
Workbench prepares and manages execution environment on HDInsight cluster using Conda. Configuration is managed by *conda_dependencies.yml* and *spark_dependencies.yml* configuration files.

You need SSH access to the HDInsight cluster in order to execute experiments in this mode.

NOTE

Supported configuration is HDInsight Spark clusters running Linux (Ubuntu with Python/PySpark 3.5.2 and Spark 2.1.11).

Overview of HDInsight-based execution for a PySpark script



Running a script on GPU

To run your scripts on GPU, you can follow the guidance in this article:[How to use GPU in Azure Machine Learning](#)

Using SSH Key-based authentication for creating and using compute targets

Azure Machine Learning Workbench allows you to create and use compute targets using SSH Key-based authentication in addition to the username/password-based scheme. You can use this capability when using remotedocker or cluster as your compute target. When you use this scheme, the Workbench creates a public/private key pair and reports back the public key. You append the public key to the `~/.ssh/authorized_keys` files for your username. Azure Machine Learning Workbench then uses ssh key-based authentication for accessing and executing on this compute target. Since the private key for the compute target is saved in the key store for the workspace, other users of the workspace can use the compute target the same way by providing the username provided for creating the compute target.

You follow these steps to use this functionality.

- Create a compute target using one of the following commands.

```
az ml computetarget attach remotedocker --name "remotevm" --address "remotevm_IP_address" --username "sshuser" --use-azureml-ssh-key
```

or

```
az ml computetarget attach remotedocker --name "remotevm" --address "remotevm_IP_address" --username "sshuser" -k
```

- Append the public key generated by the Workbench to the `~/.ssh/authorized_keys` file on the attached compute target.

IMPORTANT

You need to log on the compute target using the same username you used to create the compute target.

- You can now prepare and use the compute target using SSH key-based authentication.

```
az ml experiment prepare -c remotevm
```

Next steps

- [Create and Install Azure Machine Learning](#)
- [Model Management](#)

Create DSVM and HDI Spark cluster as compute targets

9/24/2018 • 6 minutes to read • [Edit Online](#)

NOTE

This article is deprecated. Support for earlier versions of this service will end incrementally. [View the support timeline](#). Start using the latest version with this [quickstart](#).

You can easily scale up or scale out your machine learning experiment by adding additional compute targets such as Ubuntu-based DSVM (Data Science Virtual Machine), and Apache Spark for Azure HDInsight cluster. This article walks you through the steps of creating these compute targets in Azure. For more information on Azure ML compute targets, refer to [overview of Azure Machine Learning experimentation service](#).

NOTE

You need to ensure you have proper permissions to create resources such as VM and HDI clusters in Azure before you proceed. Also both of these resources can consume many compute cores depending on your configuration. Make sure your subscription has enough capacity for the virtual CPU cores. You can always get in touch with Azure support to increase the maximum number of cores allowed in your subscription.

Create an Ubuntu DSVM in Azure portal

You can create a DSVM from Azure portal.

1. Log on to Azure portal from <https://portal.azure.com>
2. Click on the **+NEW** link, and search for "data science virtual machine for Linux".

The screenshot shows the Azure portal search interface. A search bar at the top contains the text "data science virutal machine for Linux". Below the search bar, a message says "0 results in the **Everything** category. Showing results for "data science virtual machine fo". A table below lists three virtual machine options:

NAME	PUBLISHER
Data Science Virtual Machine for Linux (Ubuntu)	Microsoft
Data Science Virtual Machine for Linux Ubuntu CSP	Microsoft
Data Science Virtual Machine for Linux (CentOS)	Microsoft

3. Choose **Data Science Virtual Machine for Linux (Ubuntu)** in the list, and follow the on-screen instructions to create the DSVM.

IMPORTANT

Make sure you choose **Password** as the *Authentication type*.

* User name

* Authentication type
SSH public key **Password**

* Password

* Confirm password

Create an Ubuntu DSVM using azure-cli

You can also use an Azure resource management template to deploy a DSVM.

NOTE

All following commands are assumed to be issued from the root folder of an Azure ML project.

First, create a `mydsvm.json` file using your favorite text editor in the `docs` folder. (If you don't have a `docs` folder in the project root folder, create one.) We use this file to configure some basic parameters for the Azure resource management template.

Copy and paste the following JSON snippet into the `mydsvm.json` file, and fill in the appropriate values:

```
{  
    "$schema": "https://schema.management.azure.com/schemas/2015-01-01/deploymentParameters.json#",  
    "contentVersion": "1.0.0.0",  
    "parameters": {  
        "adminUsername": { "value" : "<admin username>"},  
        "adminPassword": { "value" : "<admin password>"},  
        "vmName": { "value" : "<vm name>"},  
        "vmSize": { "value" : "<vm size>"}  
    }  
}
```

For the `vmSize` field, you can use any supported VM size listed in the [Ubuntu DSVM Azure resource management template](#). We recommend you use one of the below sizes as compute targets for Azure ML.

TIP

For [deep learning workloads](#) you can deploy to GPU powered VMs.

- [General Purpose VMs](#)

- Standard_DS2_v2
- Standard_DS3_v2
- Standard_DS4_v2
- Standard_DS12_v2
- Standard_DS13_v2
- Standard_DS14_v2

- [GPU powered VMs](#)

- Standard_NC6
- Standard_NC12

- o Standard_NC24

Read more about these [sizes for Linux virtual machines in Azure](#) and their [pricing information](#).

Launch CLI Window from the Azure ML Workbench app by clicking on **File --> Open Command Prompt**, or **Open PowerShell** menu item.

NOTE

You can also do this in any command-line environment where you have az-cli installed.

In the command-line window, enter the below commands:

```
# first make sure you have a valid Azure authentication token
$ az account get-access-token

# if you don't have a valid token, please log in to Azure first.
# if you already do, you can skip this step.
$ az login

# list all subscriptions you have access to
$ az account list -o table

# make sure you set the subscription you want to use to create DSVM as the current subscription
$ az account set -s <subscription name or Id>

# it is always a good idea to create a resource group for the VM and associated resources to live in.
# you can use any Azure region, but it is best to create them in the region where your Azure ML
Experimentation account is, e.g. eastus2, westcentralus or australialeast.
# also, only certain Azure regions has GPU-equipped VMs available.
$ az group create -n <resource group name> -l <azure region>

# now let's create the DSVM based on the JSON configuration file you created earlier.
# note we assume the mydsvm.json config file is placed in the "docs" sub-folder.
$ az group deployment create -g <resource group name> --template-uri
https://raw.githubusercontent.com/Azure/DataScienceVM/master/Scripts/CreateDSVM/Ubuntu/azuredeploy.json --
parameters @docs/mydsvm.json

# find the FQDN (fully qualified domain name) of the VM just created.
# you can also use IP address from the next command if FQDN is not set.
$ az vm show -g <resource group name> -n <vm name> --query "fqdns"

# find the IP address of the VM just created
$ az vm show -g <resource group name> -n <vm name> --query "publicIps"
```

Attach a DSVM compute target

Once the DSVM is created, you can now attach it to your Azure ML project.

```
# attach the DSVM compute target
# it is a good idea to use FQDN in case the IP address changes after you deallocate the VM and restart it
$ az ml computetarget attach remotedriver --name <compute target name> --address <ip address or FQDN> --
username <admin username> --password <admin password>

# prepare the Docker image on the DSVM
$ az ml experiment prepare -c <compute target name>
```

Now you should be ready to run experiments on this DSVM.

Deallocate a DSVM and restart it later

When you finish the compute tasks from Azure ML, you can deallocate the DSVM. This action shuts down the VM, releases the compute resources, but it preserves the virtual disks. You are not charged for the compute cost when the VM is deallocated.

To deallocate a VM:

```
$ az vm deallocate -g <resource group name> -n <vm name>
```

To bring the VM back to life, use the `az ml start` command:

```
$ az vm start -g <resource group name> -n <vm name>
```

Expand the DSVM OS disk

The Ubuntu DSVM comes with a 50GB OS disk and 100GB data disk. Docker stores its images on the data disk, as more space is available there. When used as compute target for Azure ML, this disk can be used up by Docker engine pulling down Docker images and building conda layers on top of it. You might need to expand the disk to a larger size (such as 200 GB) to avoid the "disk full" error while you are in the middle of an execution. Reference [How to expand virtual hard disks on a Linux VM with the Azure CLI](#) to learn how to do this easily from azure-cli.

Create an Apache Spark for Azure HDInsight cluster in Azure portal

To run scale-out Spark jobs, you need to create an Apache Spark for Azure HDInsight cluster in Azure portal.

1. Log on to Azure portal from <https://portal.azure.com>
2. Click on the **+NEW** link, and search for "HDInsight".

A screenshot of the Azure portal's search interface. The search bar at the top contains the text 'HDInsight'. Below the search bar, the word 'Results' is displayed. A table follows, showing three items: 1. 'HDInsight' published by Microsoft, 2. 'StreamSets Data Collector for HDInsight' published by StreamSets, and 3. 'HDInsight Kafka Monitoring' published by Microsoft.

NAME	PUBLISHER
HDInsight	Microsoft
StreamSets Data Collector for HDInsight	StreamSets
HDInsight Kafka Monitoring	Microsoft

3. Choose **HDInsight** in the list, and then click on the **Create** button.
4. In the **Basics** configuration screen, **Cluster type** settings, make sure you choose **Spark** as the *Cluster type*, **Linux** as the *Operating system*, and **Spark 2.1.0 (HDI 3.6)** as the *_Version*.

The screenshot shows two side-by-side configuration pages. On the left, the 'Basics' page includes fields for Cluster name (myhdiclus123.azurehdinsight.net), Subscription (BigData Demo), Cluster type (Configure required settings), Cluster login username (admin), Cluster login password, Secure Shell (SSH) username (sshuser), Resource group (Create new), and Location (West US). On the right, the 'Cluster configuration' page shows Cluster type (Spark), Operating system (Linux), Version (Spark 2.1.0 (HDI 3.6)), Cluster tier (STANDARD), and a detailed description of Spark: 'Fast data analytics and cluster computing using in-memory processing.' Below this are sections for Features (Available vs Not available) and a note about preview features.

IMPORTANT

Notice in the above screen the cluster has a *Cluster login username* field and a *Secure Shell (SSH) username* field. These are two different user identities, even though for convenience you can specify the same password for both logins. The *Cluster login username* is used to log in to the management web UI of the HDI cluster. The *SSH login username* is used to log in to the head node of the cluster, and this is what's needed for Azure ML to dispatch Spark jobs.

- Choose the cluster size and node size you need and finish the creation wizard. It can take up to 30 minutes for the cluster to finish provisioning.

Attach an HDI Spark cluster compute target

Once the Spark HDI cluster is created, you can now attach it to your Azure ML project.

```
# attach the HDI compute target
$ az ml computetarget attach cluster --name <compute target name> --address <cluster name, such as
myhdiclus123.azurehdinsight.net> --username <ssh username> --password <ssh password>

# prepare the conda environment on HDI
$ az ml experiment prepare -c <compute target name>
```

Now you should be ready to run experiments on this Spark cluster.

Next steps

Learn more about:

- [Overview of Azure Machine Learning experimentation service](#)
- [Azure Machine Learning Workbench experimentation service configuration files](#)
- [Apache Spark for Azure HDInsight cluster](#)
- [Data Science Virtual Machine](#)

How to use GPU in Azure Machine Learning

9/24/2018 • 4 minutes to read • [Edit Online](#)

NOTE

This article is deprecated. Support for earlier versions of this service will end incrementally. [View the support timeline.](#)
Start using the latest version with this [quickstart](#).

Graphical Processing Unit (GPU) is widely used to process computationally intensive tasks that can typically happen when training certain deep neural network models. By using GPUs, you can reduce the training time of the models significantly. In this document, you learn how to configure Azure ML Workbench to use [DSVM \(Data Science Virtual Machine\)](#) equipped with GPUs as execution target.

Prerequisites

- To step through this how-to guide, you need to first [install Azure ML Workbench](#).
- You need to have access to computers equipped with NVidia GPUs.
 - You can run your scripts directly on local machine (Windows or macOS) with GPUs.
 - You can also run scripts in a Docker container on a Linux machine with GPUs..

Execute in *local* Environment with GPUs

You can install Azure ML Workbench on a computer equipped with GPUs and execute against *local* environment. This can be:

- A physical Windows or macOS machine.
- A Windows VM (Virtual Machine) such as a Windows DSVM provisioned using Azure NC-series VMs template.

In this case, there are no special configuration required in Azure ML Workbench. Just make sure you have all the necessary drivers, toolkits, and GPU-enabled machine learning libraries installed locally. Simply execute against *local* environment where the Python runtime can directly access the local GPU hardware.

1. Open AML Workbench. go to **File** and **Open Command Prompt**.
2. From command line, install GPU-enabled deep learning framework such as the Microsoft Cognitive Toolkit, TensorFlow, and etc. For example:

```
REM install latest TensorFlow with GPU support
C:\MyProj> pip install tensorflow-gpu

REM install Microsoft Cognitive Toolkit 2.5 with GPU support on Windows
C:\MyProj> pip install https://cntk.ai/PythonWheel/GPU/cntk_gpu-2.5.1-cp35-cp35m-win_amd64.whl
```

1. Write Python code that leverages the deep learning libraries.
2. Choose *local* as compute environment and execute the Python code.

```
REM execute Python script in local environment
C:\MyProj> az ml experiment submit -c local my-deep-learning-script.py
```

Execute in docker Environment on Linux VM with GPUs

Azure ML Workbench also support execution in Docker in an Azure Linux VM. Here you have a great option to run computationally intensive jobs on a piece of powerful virtual hardware, and pay only for the usage by turning it off when done. While in principle it is possible to use GPUs on any Linux virtual machine, the Ubuntu-based DSVM comes with the required CUDA drivers and libraries pre-installed, making the set-up much easier. Follow the below steps:

Create a Ubuntu-based Linux Data Science Virtual Machine in Azure

1. Open your web browser and go to the [Azure portal](#)
2. Select **+ New** on the left of the portal.
3. Search for "Data Science Virtual Machine for Linux (Ubuntu)" in the marketplace.
4. Click **Create** to create an Ubuntu DSVM.
5. Fill in the **Basics** form with the required information. When selecting the location for your VM, note that GPU VMs are only available in certain Azure regions, for example, **South Central US**. See [compute products available by region](#). Click OK to save the **Basics** information.
6. Choose the size of the virtual machine. Select one of the sizes with NC-prefixed VMs, which are equipped with NVidia GPU chips. Click **View All** to see the full list as needed. Learn more about [GPU-equipped Azure VMs](#).
7. Finish the remaining settings and review the purchase information. Click Purchase to Create the VM. Take note of the IP address allocated to the virtual machine.

Create a New Project in Azure ML Workbench

You can use the *Classifying MNIST using TensorFlow* example, or the *Classifying MNIST dataset with CNTK* example.

Create a new Compute Target

Launch the command line from Azure ML Workbench. Enter the following command. Replace the placeholder text from the example below with your own values for the name, IP address, username, and password.

```
C:\MyProj> az ml computetarget attach remotedocker --name "my_dsvm" --address "my_dsvm_ip_address" --username "my_name" --password "my_password"
```

Configure Azure ML Workbench to Access GPU

Go back to the project and open **File View**, and hit the **Refresh** button. Now you see two new configuration files `my_dsvm.compute` and `my_dsvm.runconfig`.

Open the `my_dsvm.compute`. Change the `baseDockerImage` to `microsoft/mmlspark:plus-gpu-0.9.9` and add a new line `nvidiaDocker: true`. So the file should have these two lines:

```
...
baseDockerImage: microsoft/mmlspark:plus-gpu-0.9.9
nvidiaDocker: true
```

Now open `my_dsvm.runconfig`, change `Framework` value from `PySpark` to `Python`. If you don't see this line, add it, since the default value would be `PySpark`.

```
"Framework": "Python"
```

Reference Deep Learning Framework

Open `conda_dependencies.yml` file, and make sure you are using the GPU version of the deep learning framework Python packages. The sample projects list CPU versions so you need to make this change.

Example for TensorFlow:

```
name: project_environment
dependencies:
- python=3.5.2
# latest TensorFlow library with GPU support
- tensorflow-gpu
```

Example for Microsoft Cognitive Toolkit:

```
name: project_environment
dependencies:
- python=3.5.2
- pip:
  # use the Linux build of Microsoft Cognitive Toolkit 2.5 with GPU support
  - https://cntk.ai/PythonWheel/GPU/cntk_gpu-2.5.1-cp35-cp35m-win_amd64.whl
```

Execute

You are now ready to run your Python scripts. You can run them within the Azure ML Workbench using the `my_dsvm` context, or you can launch it from the command line:

```
C:\myProj> az ml experiment submit -c my_dsvm my_tensorflow_or_cntk_script.py
```

To verify that the GPU is used, examine the run output to see something like the following:

```
name: Tesla K80
major: 3 minor: 7 memoryClockRate (GHz) 0.8235
pciBusID 06c3:00:00.0
Total memory: 11.17GiB
Free memory: 11.11GiB
```

Congratulations! Your script just harnessed the power of GPU through a Docker container!

Next steps

See an example of using GPU to accelerate deep neural network training at Azure ML Gallery.

Data Source Wizard

9/24/2018 • 2 minutes to read • [Edit Online](#)

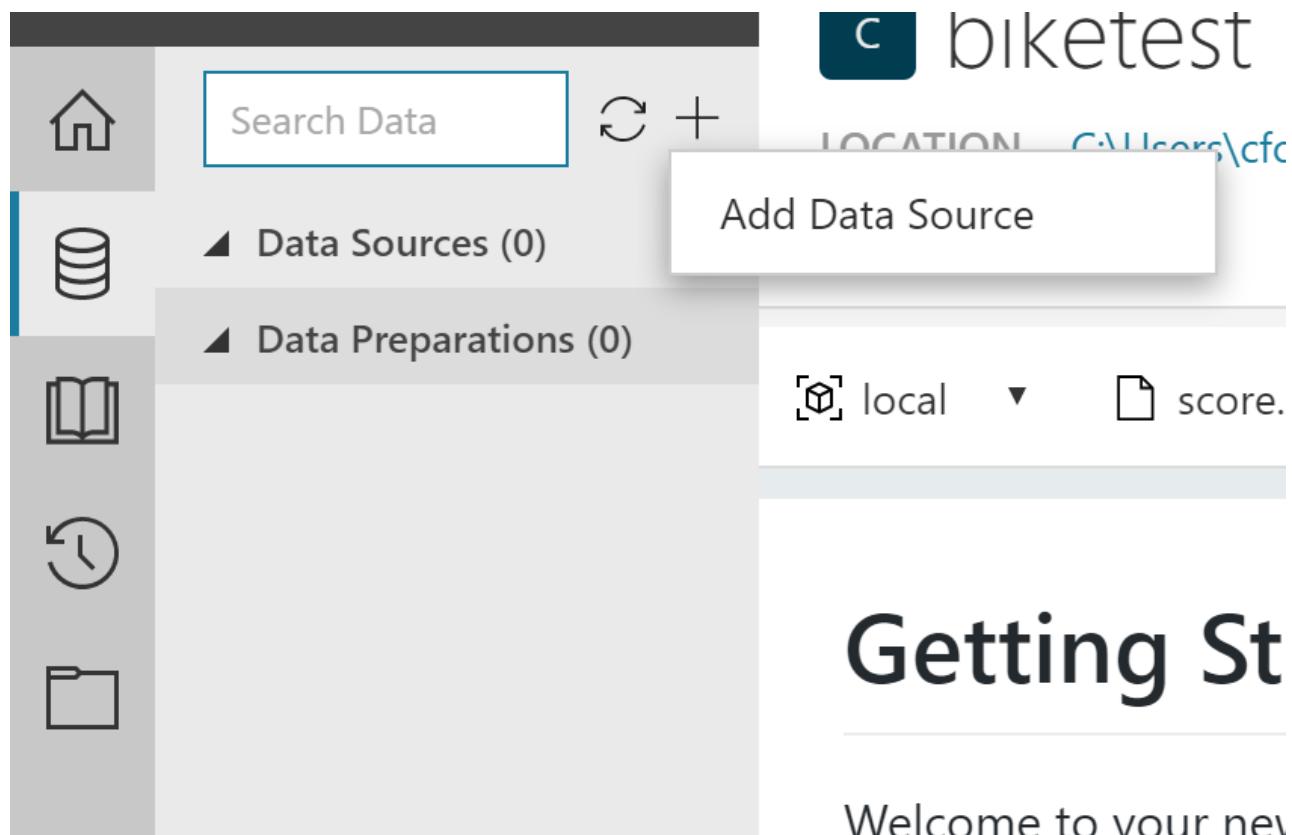
NOTE

This article is deprecated. Support for earlier versions of this service will end incrementally. [View the support timeline](#). Start using the latest version with this [quickstart](#).

The Data Source Wizard is a quick and easy way to bring a dataset into Azure ML Workbench without code. It is where you can also select a sample strategy for the dataset and data types for each column.

Step 1: Trigger the Data Source Wizard

To bring data into a project using the data source wizard. Select the + button next to the search box in the data view and choose Add Data Source.



Step 2: Select where data is stored

First, specify how your data is currently in. It could be stored in a flat file or a directory, a parquet file, an Excel file, or a database. For more information, see [Supported Data Sources](#).

Add Data Source

X

1. Data Store

Where does the data come from?

Specify the data store where the data comes from.

File(s)/Directory

Parquet

Excel

Database

Previous

Finish

Step 3: Select data file

For a file/directory, specify the file path. Choose from the dropdown the location of the data – it could be a local file path or Azure Blob Storage.

Specify the path by typing it in or clicking on the **Browse...** button to browse for it. You can browse for a directory, or one or more files.

Click **Finish** to accept the defaults for the rest of steps or Next to proceed to the next step.

Add Data Source

X

1. Data Store

Browse to find the file

Browse to the path of the file you would like to use.

Path

Local

Browse...

3. File Details

4. Data Types

5. Sampling

6. Path Column

Previous

Next

Step 4: Choose file parameters

The Data Source Wizard can automatically detect the file type, separators, and encoding. It would also show an example of what the data will look like. You can also change any of these parameters manually.

Add Data Source

Choose file parameters
Set parameters to interpret the file.

File Type: Delimited File (csv, tsv, txt, etc.)

Separator: Comma [,]

Comment Line Character:

Skip Lines Mode: Don't skip

File Encoding: utf-8

Promote Headers Mode: Use Headers From First File

	abc Path	abc tripduration	abc starttime	abc stoptime	abc start station id	abc start station ...	abc start station l...	abc start station l...	abc end station id	abc end station
1	C:\Users\cforbe..	542	2015-01-01 00...	2015-01-01 00...	115	Porter Square S..	42.387995	-71.119084	96	Cambridg..
2	C:\Users\cforbe..	438	2015-01-01 00...	2015-01-01 00...	80	MIT Stata Cent...	42.3619622	-71.0920526	95	Cambridg..
3	C:\Users\cforbe..	254	2015-01-01 00...	2015-01-01 00...	91	One Kendall Sq...	42.366277	-71.09169	68	Central Sq
4	C:\Users\cforbe..	432	2015-01-01 00...	2015-01-01 01...	115	Porter Square S..	42.387995	-71.119084	96	Cambridg..
5	C:\Users\cforbe..	735	2015-01-01 01...	2015-01-01 01...	105	Lower Cambridg..	42.356954	-71.113687	88	Inman Squ
6	C:\Users\cforbe..	311	2015-01-01 01...	2015-01-01 01...	88	Imman Square ...	42.374035	-71.101427	76	Central Sq
7	C:\Users\cforbe..	1259	2015-01-01 01...	2015-01-01 01...	91	One Kendall Sq...	42.366277	-71.09169	118	Linear Par...

Previous Next Finish

Step 5: Set data types for columns

The Data Source Wizard automatically detects the data types of the dataset's columns. If it misses one, or you wish to enforce a data type, you can manually change the data type. The **SAMPLE OUTPUT DATA** column shows you examples of what the data look like.

For columns that Data Prep infers to contain dates, you may be prompted to select the order of month and day in the date format. For example, 1/2/2013 could represent January 2nd (for this, select *Day-Month*) or February 1st (select *Month-Day*).

Add Data Source

Set the types of your data
Set the type for the columns in your data.

Show: Numeric (9) Date (2) Boolean (0) String (5)

COLUMN NAME	DATA TYPE	SAMPLE OUTPUT DATA
Path	String	C:\Users\cforbe\Desktop\201501-hubway-tripdata.csv C:\Users\cforbe\Desktop\201501-hubway-tripdata.csv C:\Users\cforbe\Desktop\201501-hubway-tripdata.csv
tripduration	Numeric	542 438 254
starttime	Date	2015-01-01 00:21:44 2015-01-01 00:27:03 2015-01-01 00:31:31
stoptime	Date	2015-01-01 00:47:47 2015-01-01 00:34:21 2015-01-01 00:35:46
start station id	Numeric	115 80 91
start station name	String	Porter Square Station MIT Stata Center at Vassar St / Main St One Kendall Square at Hampshire St / Portland St
start station latitude	Numeric	42.387995 42.3619622 42.366277
start station longitude	Numeric	-71.119084 -71.0920526 -71.09169
end station id	Numeric	96 95

Previous Next Finish

Step 6: Choose sampling strategy for data

You can specify one or more sampling strategies for the dataset, and choose one as the active strategy. The default is to load the Top 10000 rows. You can edit this sample by clicking on the **Edit** button in the toolbar or add a new

strategy by clicking on +New. The strategies that are currently support are

- Top number of rows
- Random number of rows
- Random percentage of rows
- Full file

You can specify as many sampling strategies as you want, but there is only one that can be set to active when preparing your data. You can set any strategy to be the active by selecting the strategy and click Set as Active in the toolbar.

Depending on where the data comes from, some sample strategies may not be supported. For more information about sampling, look at the sampling section in [this document](#)

Add Data Source

Data sampling
You can choose to bring in the entire file for completeness or a sample for better performance.

SAMPLE NAME	STRATEGY	DETAILS
Top 10000	Top	Count=10000
★ Full File	Full File	

1. Data Store
2. File Selection
3. File Details
4. Data Types
5. Sampling
6. Path Column

Previous Next Finish

Step 7: Path column handling

If the file path includes important data, you can choose to include it as the first column in the dataset. This option would be helpful if you are bringing in multiple files. Otherwise, you can choose to not include it.

Add Data Source X

1. Data Store

2. File Selection

3. File Details

4. Data Types

5. Sampling

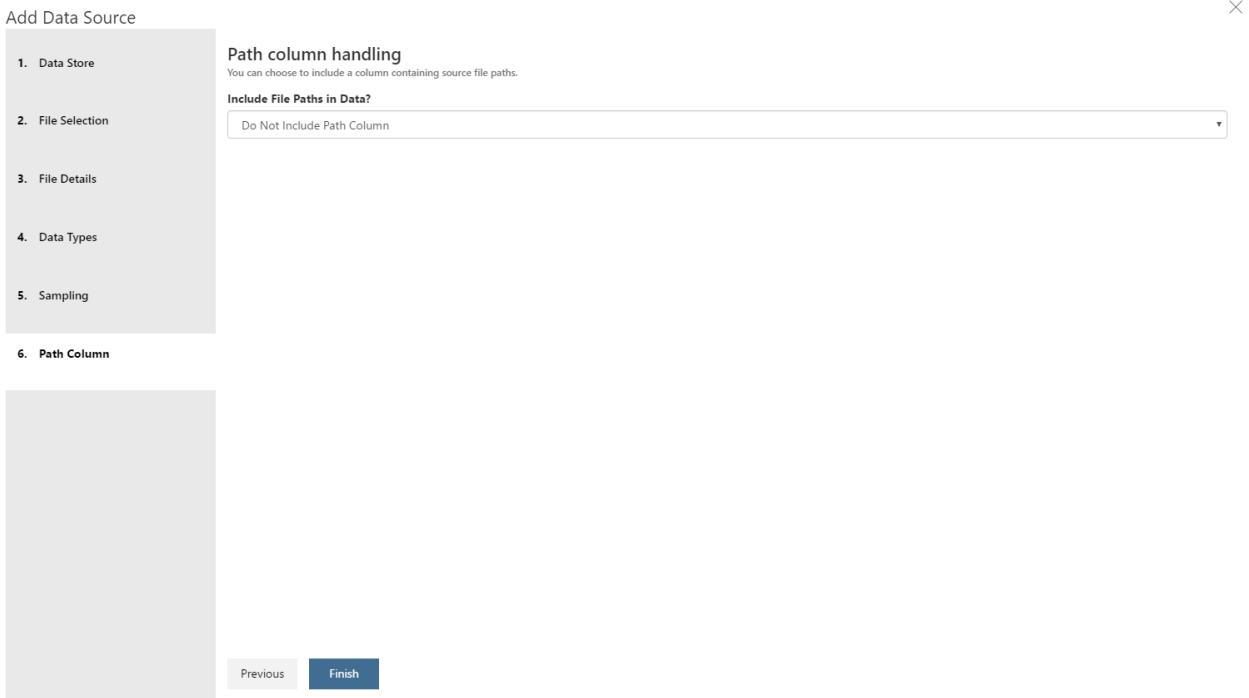
6. Path Column

Path column handling
You can choose to include a column containing source file paths.

Include File Paths in Data?

Do Not Include Path Column

[Previous](#) [Finish](#)



After clicking Finish, a new data source will be added to the project. You can find it under the Data Sources group in the Data View, or as a dsources file in the **File View**.

Getting started with data preparation

9/24/2018 • 8 minutes to read • [Edit Online](#)

NOTE

This article is deprecated. Support for earlier versions of this service will end incrementally. [View the support timeline](#). Start using the latest version with this [quickstart](#).

Welcome to the Data Preparation Getting Started Guide.

Data Preparation provides a set of tools for efficiently exploring, understanding, and fixing problems in data. It allows you to consume data in many forms and transform that data into cleansed data that is better suited for downstream usage.

Data Preparation is installed as part of the Azure Machine Learning Workbench experience. Use Data Preparation locally or deploy to a target cluster and cloud as a runtime or execution environment.

The design time runtime uses Python for extensibility and depends on various Python libraries such as Pandas. As with other components of Azure ML Workbench there is no need to install Python, it is installed for you. However if you need to install additional libraries these libraries need to be installed in the Azure ML Workbench Python directory not your usual Python directory. More details on how to install packages can be found [here](#).

A project is required before you can use Data Prep, once that project is created you may prepare data.



Navigate to the Data section of the project by selecting the Data icon on the left of the screen. Click "+" again to **Add a Data Source**. The Data Source Wizard should launch and adds a **Data Source** (.dsource) file to the project after completing the wizard. By default, the view of the data is the grid. Above the grid, it is also possible to select Metrics view. In Metrics view, summary statistics are shown. After reviewing the summary statistics, click on **Prepare** at the top of the screen. [More information about the Data Source Wizard](#)

Building blocks of data preparation

The Package

A Package is the primary container for your work. A Package is the artifact that is saved to and loaded from disk. While working inside the client, the Package is constantly AutoSaved in the background.

The output/results of a Package can be explored in Python or via a Jupyter Notebook.

A Package can be executed across multiple runtimes including local Python, Spark (including in Docker), and HDInsight.

A Package contains one or more Dataflows that are the steps and transforms applied to the data.

A Package may use another Package as a Data Source (referred to as a Reference Data Flow).

The dataflow

A Dataflow has a source and optional Transforms which are arranged through a series of Steps and optional destinations. Clicking on a Step re-executes all the sources and Transforms prior to and including the selected Step. The transformed data through that step is shown within the grid. Steps can be added, moved, and deleted within a Dataflow through the Step List.

The Step List on the right side of the client can be opened and closed to provide more screen space.

Multiple Dataflows can exist in the UI at a time, each Dataflow is represented as a tab in the UI.

The source

A source is where the data comes from, and the format it is in. A Data Prep Package always sources its data from another Data Flow(Data Source). It is this reference Data Flow that contains the information. Each source has a different user experience to allow it to be configured. The source produces a "rectangular"/tabular view of the data. If the source data originally has a "ragged right", then the structure is normalized to be "rectangular." [Appendix 2 provides the current list of supported sources.](#)

The transform

Transforms consume data in a given format, perform some operation on the data (such as changing the data type) and then produce data in the new format. Each Transform has its own UI and behavior(s). Chaining several Transforms together via Steps in the Dataflow is the core of Data Preparation functionality. [Appendix 3 provides the current list of supported Transforms.](#)

The inspector

Inspectors are visualizations of the data and are available to improve understanding of the data. Understanding the data and data quality issues helps you decide which actions (Transforms) should be taken. Some Inspectors support actions that generate Transforms. For example, the Value Count Inspector allows you to select a Value and then apply a filter to include that Value or to Exclude that Value. Inspectors can also provide context for Transforms. For example, selecting one or more columns changes the possible Transforms that can be applied.

A column may have multiple Inspectors at any point in time (for example, Column Statistics and a Histogram). There can also be instances of an Inspector across multiple columns. For example, all numeric columns could have Histograms at the same time.

Inspectors appear in the Profiling Well at the bottom of the screen. Maximize inspectors to see them larger within the main content area. Think of the data grid as the default inspector. Any Inspector can be expanded into the main content area. Inspectors within the main content area minimize to the Profiling Well. Customize an Inspector by clicking on the pencil icon within the Inspector. Reorder Inspectors within the Well using drag and drop.

Some Inspectors support a "Halo" mode. This mode shows the value or state before the last Transform was applied. The old value is displayed in gray with the current value in the foreground and shows the impact of a Transform. [Appendix 4 provides the current list of supported Inspectors.](#)

The destination

A Destination is where you write/export the data to after you have prepared it in a Dataflow. A given Dataflow can have multiple Destinations. Each Destination has a different user experience to allow it to be configured. The Destination consumes data in a "rectangular"/tabular format and writes it out to some location in a given format. [Appendix 5 provides the current list of supported Destinations.](#)

Using data preparation

Data Preparation assumes a basic five-step methodology/approach to data preparation.

Step 1: Ingestion

Import data for Data Preparation by using the **Add Data Source** option within the project view. All initial ingestion of data is handled through the Data Source Wizard.

Step 2: Understand/profile the data

First, look at the Data Quality Bar at the top of each column. Green represents the rows that have values. Gray represents rows with a missing value, null, etc. Red indicates error values. Hover over the bar to get a tool tip with the exact numbers of rows in each of the three buckets. The Data Quality Bar uses a logarithmic scale so always check the actual numbers to get a rough feel for the volume of missing data.

The screenshot shows the Azure Machine Learning Workbench Dataflows interface. On the left, there's a sidebar with icons for Home, Data Sources, and Data Preparations. The main area is titled "DATAFLOWS" and shows a "Join Result" between "201701-hubway-tripdata" and "BostonWeather". The "BostonWeather" data source is highlighted. To the right is a grid table with columns: DATE, # HOURLYD..., # HOURLYRe..., and # HOURLYW... The table has 6 rows of data. A status bar at the top right indicates "Valid: 18938 Missing: 0 Error: 5".

Next, use a combination of other Inspectors plus the grid to better understand data characteristics. Start formulating hypotheses about the data preparation required for further analysis. Most inspectors work on a single column or a small number of columns.

It's likely that several Inspectors across several columns are needed to understand the data. You can scroll through various Inspectors in the Profiling Well. Within the well, you can also move Inspectors to the head of the list in order to see them in the immediately viewable area.

This screenshot shows the same Dataflows interface with the Profiling Well open on the right side. The "STEPS" list contains various data processing steps like Reference dataflow, Sample, Filter REPORTTYPE, Remove, Copy, Handle error values in HOURLYDRYBULBTM, Delete column by example, Rename Column to Hour Range, Remove DATE, Summarize, and Transform dataflow. Below the steps, two inspectors are visible: a Histogram for "N_RelativeHumidity" and a Box Plot for "N_DryBulbTemp". The histogram shows a bell-shaped curve over a series of bars. The box plot shows the distribution of "N_DryBulbTemp" with a median around 500.

Different Inspectors are provided for continuous vs categorical variables/columns. The Inspector menu enables and disables options depending on the type of variables/columns you have.

When working with wide datasets that have many columns, a pragmatic approach of working with subsets is advisable. This approach includes focusing on a small number of columns (for example, 5-10), preparing them and then working through the remaining columns. The grid inspector supports vertical partitioning of columns and so if you have more than 300 columns then you need to "page" through them.

Step 3: Transform the data

Transforms change the data and allow the execution of the data to support the current working hypothesis.

Transforms appear as Steps in the Step List on the right-hand side. It is possible to "time travel" through the Step List by clicking on any arbitrary point in the Step List.

A green icon to the left of a given Step indicates that it has run and the data reflects the execution of the Transform. A vertical bar to the left of the Step indicates the current state of the data in the Inspectors.

STEPS ○ >>

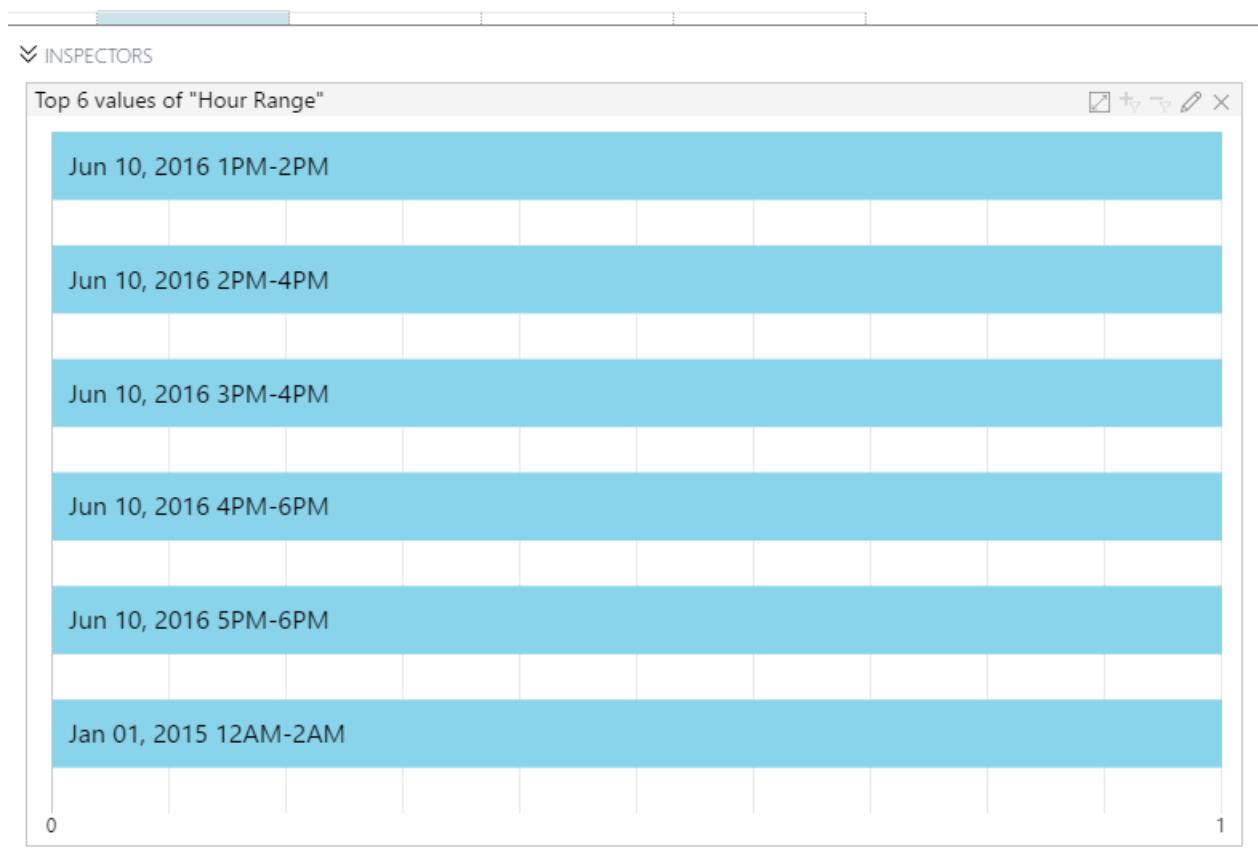
- ✓ Reference dataflow
- ✓ Sample
- ✓ Filter REPORTTYPE
- ✓ Remove REPORTTYPE
- ✓ Convert HOURLYDRYBULBTEMPF
- ✓ Handle error values in HOURLYDRYBULBTEMPF
- ✓ Derive Column by example
- ✓ Rename Column to Hour Range
- ✓ Remove DATE
- Summarize
- Transform dataflow

Try to make small frequent changes to the data and to validate (Step 4) after each change as the hypothesis evolves.

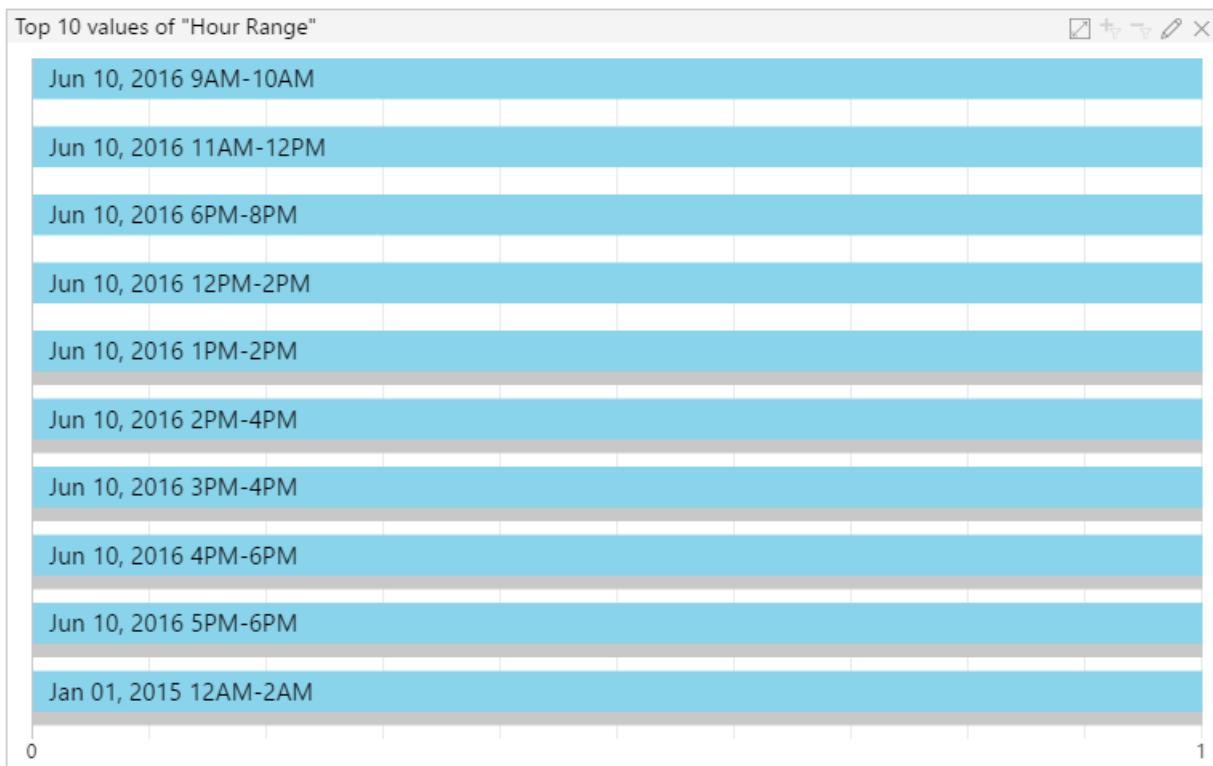
Step 4: Verify the impact of the transformation.

Decide if the hypothesis was correct. If correct, then develop the next hypothesis and repeat steps 2-3 for the new one. If incorrect, then undo the last transformation and develop a new hypothesis and repeat steps 2-3.

The primary way to determine if the Transform had the right impact is to use the Inspectors. Use existing. Use Inspectors with the Halo effect enabled or launch multiple Inspectors to view the data at given points in time.



▼ INSPECTORS



To undo a Transformation, go the Steps List on the right-hand side of the UI. (The Steps List panel may need to be popped back out. To open it, click the double chevron pointing left). In the panel, select the Transform that was executed that you wish to undo. Select the drop-down on the right-hand side of the UI block. Select either **Edit** to make changes or **Delete** to remove the Transform from the Steps List and the Dataflow.

Step 5: Output

When finished with your data preparation, you can write the Dataflow to an output. A Dataflow can have many outputs. From the Transforms menu, you can select which output you want the dataset to be written as. You can also select the output's destination.

List of Appendices

- [Appendix 2 - Supported Data Sources](#)
- [Appendix 3 - Supported Transforms](#)
- [Appendix 4 - Supported Inspectors](#)
- [Appendix 5 - Supported Destinations](#)
- [Appendix 6 - Sample Filter Expressions in Python](#)
- [Appendix 7 - Sample Transform Dataflow Expressions in Python](#)
- [Appendix 8 - Sample Data Sources in Python](#)
- [Appendix 9 - Sample Destination Connections in Python](#)
- [Appendix 10 - Sample Column Transforms in Python](#)

See Also

- [Advanced data preparation tutorial](#)

Data Preparations user guide

9/24/2018 • 5 minutes to read • [Edit Online](#)

NOTE

This article is deprecated. Support for earlier versions of this service will end incrementally. [View the support timeline](#). Start using the latest version with this [quickstart](#).

The Azure Machine Learning Data Preparations experience provides a lot of rich functionality. This article documents the deepest parts of the experience.

Step execution, history, and caching

Data Preparations step history maintains a series of caches for performance reasons. If you select a step and it hits a cache, it doesn't re-execute. If you have a write block at the end of the step history and you flip back and forth on the steps but make no changes, the write isn't triggered after the first time. A new write occurs and overwrites the previous one, if you:

- Make changes to the write block.
- Add a new transform block and move it above the write block, which generates a cache invalidation.
- Change the properties of a block above the write block, which generates a cache invalidation.
- Select refresh on a sample (thus invalidating all the caches).

Error values

Data transformations might fail for an input value because that value can't be handled appropriately. For example, in the case of type coercion operations, the coercion fails if the input string value can't be cast to the specified target type. A type coercion operation might be converting a column of string type to a numeric or Boolean type or attempting to duplicate a column that doesn't exist. (This failure occurs as the result of moving the *delete column X* operation before the *duplicate column X* operation.)

In these cases, Data Preparations produces an error value as the output. Error values indicate that a previous operation failed for the given value. Internally, they're treated as a first-class value type, but their presence doesn't alter the underlying type of a column, even if a column consists entirely of error values.

Error values are easy to identify. They're highlighted in red and read "Error." To determine the reason for the error, hover over an error value to see a text description for the failure.

Error values propagate. After an error value occurs, it propagates in most cases as an error through most operations. There are three ways to replace or remove them:

- Replace
 - Right-click a column, and select **Replace Error Values**. You can then choose a replacement value for each error value found in the column.
- Remove
 - Data Preparations includes interactive filters to preserve or remove error values.
 - Right-click a column, and select **Filter Column**. To preserve or remove error values, create a conditional with the condition "*is error*" or "*is not error*".
- Use a Python expression to conditionally operate on error values. For more information, see the [section on Python extensions](#).

Sampling

A Data Sources file takes in raw data from one or more sources, either from the local file system or a remote location. The Sample block allows you to specify whether to work with a subset of the data by generating samples. Operating on a sample of the data rather than a large dataset often leads to better performance when you carry out operations in later steps.

For each Data Sources file, multiple samples can be generated and stored. However, only one sample can be set as the active sample. You can create, edit, or delete samples in the Data Source wizard or by editing the Sample block. Any Data Preparations files that reference a data source inherently use the sample specified in the Data Sources file.

There are a number of sampling strategies available, each with different configurable parameters.

Top

This strategy can be applied to either local or remote files. It takes the first N rows (specified by Count) into the data source.

Random N

This strategy can be applied only to local files. It takes random N rows (specified by Count) into the data source. You can provide a specific seed to ensure that the same sample is generated, provided that Count is also the same.

Random %

This strategy can be applied to either local or remote files. In both cases, a probability and a seed must be provided, similar to the Random N strategy.

For samples of remote files, additional parameters need to be provided:

- Sample generator
 - Select a Spark cluster or remote Docker compute target to be used for the sample generation. The compute target must be created for the project beforehand for it to appear in this list. Follow the steps in the section "Create a new compute target" in [How to use GPU in Azure Machine Learning](#) to create compute targets.
- Sample storage
 - Provide an intermediate storage location to store the remote sample. This path must be a different directory from the input file location.

Full file

This strategy can be applied only to local files, taking the full file into the data source. If the file is too large, this option might slow down future operations in the app. You might find it more appropriate to use a different sampling strategy.

Fork, merge, and append

When you apply a filter over a dataset, the operation splits the data into two result sets: one set represents records that succeed in the filter, and another set is for the records that fail. In either case, the user can choose which result set to display. The user can discard the other dataset or place it in a new data flow. The latter option is referred to as forking.

To fork:

1. Select a column, right-click, and select the **Filter** column.
2. Under **I Want To**, select **Keep Rows** to display the result set that passes the filter.
3. Select **Remove Rows** to display the failed set.
4. After **Conditions**, select **Create Dataflow Containing the Filtered Out Rows** to fork the non-display result set into a new data flow.

This practice is often used to separate out a set of data that requires additional preparation. After you prepared the forked dataset, it's common to merge the data with the result set in the original data flow. To perform a merge (the reverse of a fork operation), use one of the following actions:

- **Append Rows.** Merge two or more data flows vertically (row-wise).
- **Append Columns.** Merge two or more data flows horizontally (column-wise).

NOTE

Append Columns fails if a column collision occurs.

After a merge operation, one or more data flows are referenced by a source data flow. Data Preparations notifies you with a notification in the lower-right corner of the app, beneath the list of steps.

Any operation on the referenced data flow requires the parent data flow to refresh the sample used from the referenced data flow. In that event, a confirmation dialog box replaces the data flow reference notification in the lower-right corner. That dialog box confirms that you need to refresh the data flow to synchronize with changes to any dependency data flows.

List of appendices

- [Supported data sources](#)
- [Supported transforms](#)
- [Supported inspectors](#)
- [Supported destinations](#)
- [Sample filter expressions in Python](#)
- [Sample transform data flow expressions in Python](#)
- [Sample data sources in Python](#)
- [Sample destination connections in Python](#)
- [Sample column transforms in Python](#)

Use data transforms for data preparation in Azure Machine Learning

9/24/2018 • 11 minutes to read • [Edit Online](#)

NOTE

This article is deprecated. Support for earlier versions of this service will end incrementally. [View the support timeline](#). Start using the latest version with this [quickstart](#).

A *transform* in Azure Machine Learning consumes data in a given format, performs an operation on the data (such as changing the data type), and then produces data in the new format. Each transform has its own interface and behavior. By chaining several transforms together via steps in the data flow, you can perform complex and repeatable transformations on your data. This is the core of data preparation functionality.

The following are the transforms available in Azure Machine Learning.

Column selection

Many of the transforms in this list work on either a single column or many. To select multiple columns, use the Ctrl key. To select a range of columns, use the Shift key.

Transforms from the main menu or the grid header

Access transforms from the Transforms option on the main menu. You can also select transforms by right-clicking the column name in the data grid. If multiple columns are selected, right-clicking any of them provides a transforms menu.

The right-click menu only shows valid transforms for the data type selected. The main menu offers all transforms but disables those not relevant to the selected columns.

A small subset of contextual transforms is available by right-clicking a cell. These transforms are Copy, Replace, and Filter. These are data type-aware, so the options for a number column are different than for a string column.

Derive Column by Example

Use this transform to create a new column as a derivative of one or more existing columns. The transform looks at the input (selected) columns and the example given, and then determines the desired output in the new column.

To use this transform, select one or more columns. Add a new (blank) derived column by example. Type an example of what you want to see in the derived column (assuming it's derived from other columns), and the "By Example" technology attempts to fill in all the other cells in the column.

For complicated examples, you might need to provide more than one example. To do this, select another cell and type another example.

The "By Example" technology uses selected columns to attempt to determine the meaning of an example. If no columns are selected when this transform is invoked, then all cells for the current row are used. Selecting only the required columns leads to more accurate results.

You can select columns before invoking the transform. When the transform editor has been opened, check boxes at the top of each column indicate what columns are selected as inputs. You can add or remove columns from the

selection by using the check boxes in the column headers.

For a more detailed explanation of the **Derived Column by Example** transform, along with more samples, see [Derive Column by Example reference](#).

Split Column by Example

This transform takes an existing column and, by using the “By Example” engine, attempts to split that column into n other columns. You can run the auto-split on the subsequent generated columns.

For a more detailed explanation of the **Split Column by Example** transform, along with more samples, see [Split Column by Example reference](#).

Expand JSON

This transform enables you to add multiple columns by expanding a column with valid JSON text.

For a more detailed explanation of the **Expand JSON** transform, along with more samples, see [Expand JSON reference](#).

Combine Columns by Example

This transform adds a new column by combining values from multiple columns.

For a more detailed explanation of the **Combine Columns by Example** transform, along with more samples, see [Combine Columns by Example reference](#).

Duplicate Column

This transform makes an exact copy of one or more selected columns and gives each a new name derived from the original column name.

Text Clustering

This transform is designed to take inconsistent values that should be the same and group them together.

With this transform, the values in a single column are analyzed for similarity and grouped into clusters. For each cluster, there is a canonical value, which is the value that replaces all instances in the cluster and all instance values. Complete clusters can be removed and the canonical value edited. Instances can be removed from a given cluster. The similarity score threshold filter that's used to group instances into a cluster can be changed.

By default, this transform replaces all cluster instance values with the canonical value for that cluster, creating a new column to contain the new values. You can also add the similarity score for each instance to a new column (that can be named) for use later in the data flow.

Replace Values

Use this transform to replace one string with another. The source string can be a partial string or a full cell, and the replacement can apply to a single column or many. The search string supports searching for special characters as well as for regular characters.

Replace NA Values

Use this transform to replace the various forms of NA (N/A, NA, null, NaN, etc.), or to replace empty strings with a single value to make them consistent. This transform supports one or many columns, and it's only listed when a column is selected. It's not present on the main transform menu when no columns are selected.

Replace Missing Values

This transform replaces missing data with a single value, and it supports one or many columns. This transform is only listed when a column is selected. It's not present on the main transform menu when no columns are selected.

Replace Error Values

This transform replaces errors with a single value, and it supports one or many columns. This transform is only listed when a column is selected. It's not present on the main transform menu when no columns are selected.

Trim String

This transform removes leading and trailing "whitespace" characters (including spaces, tabs, etc.) from one or more columns.

Adjust Precision

This transform sets the number of decimal places for a numeric column.

Rename Column

This transform changes the name of the selected column. You can also invoke it inline in the column header by clicking the name of the column.

Remove Column

This transform removes the selected columns, and it works on a single column or many.

Keep Column

This transform keeps only the selected columns, and it works on a single column or many.

Handle Path Column

During the import of a file, a path column is automatically added to the data set by the **Add Data Source** feature. It contains the fully qualified filename that forms the path to the data set. This transform either adds or removes that extra column from the data set.

Convert Field Type to Numeric

This transform changes the column type to numeric. You can specify the separator for non-integer data. By default, this transform doesn't prompt for the separator, so use the **Edit** menu item to invoke the editor. This transform works on a single column or many.

Convert Field Type to Date

This transform changes the column type to date. A default date/time format is used, but it can be overridden by using `strftime` directives. You can also prepend time values with a fixed date.

By default, this transform doesn't prompt for the format, so use the **Edit** menu item on the resultant step to invoke the editor. This transform works on a single column or many.

Only dates between 9-22-1677 and 4-11-2262 can be converted to the date type.

Convert Field Type to Boolean

This transform changes the column type to true/false. It supports mapping multiple values to either true or false, and you can edit these mappings. It also supports a constant to which you can map values that don't exist in the true/false mapping tables. This transform works on a single column or many.

Convert Field Type to String

This transform changes the column type to string, and it works on a single column or many.

Convert Unix Timestamp to DateTime

This transform understands the Unix timestamp format, even if it's represented as a string in the data. It converts the timestamp correctly to a date type in data preparation.

Filter

This transform supports filtering of rows based on the values in one or many columns. The conditions of the filter depend on the data type of each column, so you can filter string columns by "contains" or "does not contain." Numeric columns can be filtered by "greater than" or "less than" conditions.

When the **Filter** transform is invoked from the main menu, or from right-clicking the header of a column, the option to fork the failing rows into another data flow is available. The main data flow continues with filtered **in** rows, and a new data flow is created that contains all the rows that were filtered **out**.

Use First Row as Headers

This transform promotes the first row from the data set to be the column names. If some columns don't have data in the first row, a name is auto-generated. Using this transform means that the import of data starts at row 2 at the earliest. Depending on the **Skip Rows** setting, the import can start even further down in the data set. You can also use it to remove the promotion and to use auto-generated names only.

Join

This transform is used to join two data flows together. You can select which output of the join to be the result: the successful rows from the join, the "left" failing rows from the join, or the "right" failing rows from the join.

The **Join** transform starts from a single data flow and selects that data flow as one side of the join. It then prompts you to select another data flow for the other side of the join. After you've selected the two flows, the transform requires a single column on each side of the join to be selected to join on. If the join needs more than one column, create a derived column before starting the transform to create a new, concatenated column to be used only for the join. The transform attempts to suggest join keys and, if possible, generate the derived column automatically.

After the join has been completed, a Sankey diagram view of the join is presented. The width of the lines relative to each other represents the number of rows moving through that part of the join flow. Using the panel on the right, you can select the successful rows, the left failing rows, or the right failing rows exclusively. You can also choose a single branch.

Append Rows

This transform appends data from another data flow to the current one. It maps the columns by position to add the new rows at the end.

Append Columns

This transform appends new columns from another data flow to the current one. It adds the new columns to the right. It doesn't attempt to line up data in rows.

Summarize

This transform computes aggregates for the combination of unique values in one or more selected columns.

The aggregates supported are: COUNT, SUM, MIN, MAX, MEAN, VARIANCE, and STANDARD DEVIATION. The list of aggregates for a given column is filtered to only the aggregates that apply to the data type. Aggregates that do not apply are disabled. For example, it's not possible to compute the MEAN of a string column, but it's possible to compute the MIN and the MAX.

When the editor is available, drag from the column header up into the panel at the top left, where the columns to be aggregated are displayed. This panel is hierarchical, so you can do nested aggregates. The editor panel at the top right is used to select which aggregate to apply to a column. A single column can be aggregated one or more times. After at least one aggregate has been chosen, the grid at the bottom right previews the data in its aggregate form.

This transform is analogous to an [ANSI-SQL GROUP BY](#).

Remove Duplicates

This transform removes the entire row when there are duplicate values in one or more selected columns. If no columns are chosen, the only rows removed are ones where all the column values are the same.

Sort

This transform sorts the data. The sort can be done by a single column or many, and each column can be sorted ascending (the default) or descending (which can be changed from the editor).

This transform is analogous to an [ANSI-SQL ORDER BY](#).

Output transforms

The following transforms output data. You can have multiple write blocks in a single flow to write out the data at different points.

Write to CSV

This transform writes out the data in CSV form from the current point in the data flow. It controls the location (local or remote) and various settings around the file.

Write to Parquet

This transform writes out the data in Parquet form from the current point in the data flow. It controls the location (local or remote) and various settings around the file.

Script-based transforms

The following transforms use script (Python) to perform functionality that's missing in the core product.

NOTE

Before you use any of these transforms, read [Using Python extensibility](#).

Add Column (script)

This transform adds a column to the data using a Python expression. For more information, see [Sample Python](#)

[code for deriving new columns.](#)

Advanced Filter (script)

Use this transform to write a Python row level filter. For more information, see [Example filter expressions](#).

Transform Dataflow (script)

This transform applies Python to the entire data set. For more information, see [Example transform data flow transformations](#).

This transform also can apply Python to an entire data partition. For more information, see [Example transform data flow transformations](#).

Write Dataflow (script)

This transform uses Python to write out an entire data set. For more information, see [Sample Python for deriving new columns](#).

Combine columns by example transformation

9/24/2018 • 3 minutes to read • [Edit Online](#)

NOTE

This article is deprecated. Support for earlier versions of this service will end incrementally. [View the support timeline](#). Start using the latest version with this [quickstart](#).

This transformation allows user to add a new column by combining values from multiple columns. User can specify a separator or provide examples of combined values to perform this transform. When the user provides examples of combination, the transformation is handled by the same **By-Example** engine that is used in the **Derive Column by Example** transform.

How to perform this transformation

To perform this transform, follow these steps:

1. Select two or more columns that you want to combine into one column.
2. Select **Combine Columns by Example** from the **Transforms** menu. Or, Right click on the header of any of the selected columns and select **Combine Columns by Example** from the context menu. The Transform Editor opens, and a new column is added next to the right most selected column. The new column contains the combined values separated by a default separator. Selected columns can be identified by the checkboxes at the column headers. Addition and removal of columns from the selection can be done using the checkboxes.
3. You can update the combined value in the newly created column. The updated value is used as an example to learn the transform.
4. Click **OK** to accept the transform.

Transform editor: Advanced mode

Advanced Mode provides a richer experience for Combining columns.

Selecting **Separator** under **Combine Columns by** enables user to specify Strings in the **Separator** text box. Tab out from the **Separator** text box to preview the results in the data grid. Press **OK** to commit the transform.

Selecting **Examples** under **Combine Columns by** enables user to provide examples of combined values. To promote a row as an example, double-click on the rows in the grid. Type in the expected output in the text box against the promoted row. Tab out from the **Separator** text box to preview the results in the data grid. Press **OK** to commit the transform.

User can switch between the **Basic Mode** and the **Advanced Mode** by clicking the links in the Transform Editor.

Transform editor: Send Feedback

Clicking on the **Send feedback** link opens the **Feedback** dialog with the comments box prepopulated with the examples user has provided. User should review the content of the comments box and provide more details to help us understand the issue. If the user does not want to share data with Microsoft, user should delete the prepopulated example data before clicking the **Send Feedback** button.

Editing existing transformation

A user can edit an existing **Combine Column By Example** transform by selecting **Edit** option of the Transformation Step. Clicking on **Edit** opens the Transform Editor in **Basic Mode**. User can enter the **Advanced Mode** by clicking on the link in the header. All the examples that were provided during creation of the transform

are shown there.

Example using separators

A comma followed by a space is used as a separator in this example to combine the *Street*, *City*, *State*, and *ZIP* columns.

STREET	CITY	STATE	ZIP	COLUMN
16011 N.E. 36th Way	REDMOND	WA	98052	16011 N.E. 36th Way, REDMOND, WA, 98052
16021 N.E. 36th Way	REDMOND	WA	98052	16021 N.E. 36th Way, REDMOND, WA, 98052
16031 N.E. 36th Way	REDMOND	WA	98052	16031 N.E. 36th Way, REDMOND, WA, 98052
16041 N.E. 36th Way	REDMOND	WA	98052	16041 N.E. 36th Way, REDMOND, WA, 98052
16051 N.E. 36th Way	REDMOND	WA	98052	16051 N.E. 36th Way, REDMOND, WA, 98052
16061 N.E. 36th Way	REDMOND	WA	98052	16061 N.E. 36th Way, REDMOND, WA, 98052
3460 157th Avenue NE	REDMOND	WA	98052	3460 157th Avenue NE, REDMOND, WA, 98052
3350 157th Ave N.E.	REDMOND	WA	98052	3350 157th Ave N.E., REDMOND, WA, 98052
3240 157th Avenue N.E.	REDMOND	WA	98052	3240 157th Avenue N.E., REDMOND, WA, 98052

Example using By example

The value in **bold** was provided as an example.

DATE	MONTH	YEAR	HOUR	MINUTE	SECOND	COMBINED COLUMN
13	Oct	2016	15	01	23	13-Oct-2016 15:01:23 PDT

DATE	MONTH	YEAR	HOUR	MINUTE	SECOND	COMBINED COLUMN
16	Oct	2016	16	22	33	16-Oct-2016 15:01:33 PDT
17	Oct	2016	12	43	12	17-Oct-2016 15:01:12 PDT
12	Nov	2016	14	22	44	12-Nov-2016 15:01:44 PDT
23	Nov	2016	01	52	45	23-Nov-2016 15:01:45 PDT
16	Jan	2017	22	34	56	16-Jan-2016 15:01:56 PDT
23	Mar	2017	01	55	25	23-Mar-2016 15:01:25 PDT
16	Apr	2017	11	34	36	16-Apr-2016 15:01:36 PDT

Derive column by example transformation

9/24/2018 • 12 minutes to read • [Edit Online](#)

NOTE

This article is deprecated. Support for earlier versions of this service will end incrementally. [View the support timeline](#). Start using the latest version with this [quickstart](#).

The **Derive Column by Example** transform enables users to create a derivative of one or more existing columns using user provided examples of the derived result. The derivative can be any combination of the supported String, Date, and Number transformations.

Following String, Date, and Number transformations are supported:

String transformations:

Substring including intelligent extraction of Number and Dates, Concatenation, Case Manipulation, Mapping Constant Values.

Date transformations:

Date Format change, Extracting Date Parts, Mapping Time to Time Bins.

The date transformations are fairly generic with a few notable limitations:

- Timezones are unsupported.
- Some common formats that are unsupported:
 - ISO 8601 week of year format (for example "2009-W53-7")
 - Unix epoch time.
- All formats are case-sensitive (notably "4am" is not recognized as a time although "4AM" is).

Number transformations:

Round, Floor, Ceiling, Binning, Padding with zeros or space, Division or Multiplication by a power of 1000.

Composite transformations:

Any combination of String, Number, or Date Transformations.

How to use this transformation

To perform this transform, follow these steps:

1. Select one or more columns that you want to derive the value from.
2. Select **Derive Column by Example** from the **Transforms** menu. Or, Right click on the header of any of the selected columns and select **Derive Column by Example** from the context menu. The Transform Editor opens and a new column is added next to the right most selected column. Selected columns can be identified by the checkboxes in the column headers. Addition and removal of columns from the selection can be done by using the checkboxes in the column headers.
3. Type an example of the *output* against a row, and press enter. At this point, the Workbench analyzes the input column as well as the provided output to synthesize a program that can transform the given inputs into output. The synthesized program is executed against all the rows in the data grid. For ambiguous and complicated cases, multiple examples may be needed. Depending on whether you are in Basic Mode or Advanced Mode,

- multiple examples can be provided in different ways.
- Review the output and Click **OK** to accept the transform.

Transform editor: basic mode

Basic Mode provides an inline editing experience in the data grid. You can provide examples of the output by navigating to the cell of interest and typing the value.

The workbench analyses the data and tries to identify the edge cases that should be reviewed by the user. While the data is being analyzed, **Analyzing Data** is shown in the header of the Transform Editor. Once the analysis is complete, either **No Suggestions** or, **Review next suggested row** is displayed in the header. You can navigate through the edge cases by clicking on **Review next suggested row**. In case the value is incorrect for a row, you should key in the correct value as additional example.

Transform editor: advanced mode

Advanced Mode provides a richer experience for Deriving columns by example. All the examples are shown at one place. You can also review all the edge cases at one place by clicking on **Show suggested examples**.

In the advanced mode, you can add any row as an example row by double-clicking on the row in the grid. Once a row is copied as an example row, you can also edit the data in the source columns to make a synthetic example. By doing so, you can add cases that are not currently present in the sample data.

User can switch between the **Basic Mode** and the **Advanced Mode** by clicking the links in the Transform Editor.

Transform editor: Send Feedback

Clicking on the **Send feedback** link opens the **Feedback** dialog with the comments box prepopulated with the examples user has provided. User should review the content of the comments box and provide more details to help us understand the issue. If the user does not want to share data with Microsoft, user should delete the prepopulated example data before clicking the **Send Feedback** button.

Editing existing transformation

A user can edit an existing **Derive Column By Example** transform by selecting **Edit** option of the Transformation Step. Clicking on **Edit** opens the Transform Editor in **Advanced Mode**, and all the examples that were provided during creation of the transform are shown.

Examples of string transformations by example

NOTE

Values in **bold** represent the examples that were provided in order to complete the transformation in the shown dataset.

S1. Extracting file names from file paths

Number of Examples that were required for this case: 2

INPUT	OUTPUT
C:\Python35\Tools\pynche\TypeinViewer.py	TypeinViewer.py
C:\Python35\Tools\pynche\webcolors.txt	webcolors.txt
C:\Python35\Tools\pynche\websafe.txt	websafe.txt
C:\Python35\Tools\pynche\X\rgb.txt	rgb.txt

INPUT	OUTPUT
C:\Python35\Tools\pynche\x\xlicense.txt	xlicense.txt
C:\Python35\Tools\Scripts\2to3.py	2to3.py
C:\Python35\Tools\Scripts\analyze_dxp.py	analyze_dxp.py
C:\Python35\Tools\Scripts\byext.py	byext.py
C:\Python35\Tools\Scripts\byteyears.py	byteyears.py
C:\Python35\Tools\Scripts\checkappend.py	checkappend.py

S2. Case manipulation during string extraction

Number of Examples that were required for this case: 3

INPUT	OUTPUT
REINDEER CT & DEAD END; NEW HANOVER; Station 332; 2015-12-10 @ 17:10:52;	New Hanover
BRIAR PATH & WHITEMARSH LN; HATFIELD TOWNSHIP; Station 345; 2015-12-10 @ 17:29:21;	Hatfield Township
HAWS AVE; NORRISTOWN; 2015-12-10 @ 14:39:21- Station:STA27;	Norristown
AIRY ST & SWEDE ST; NORRISTOWN; Station 308A; 2015-12-10 @ 16:47:36;	Norristown
CHERRYWOOD CT & DEAD END; LOWER POTTSGROVE; Station 329; 2015-12-10 @ 16:56:52;	Lower Pottsgrove
CANNON AVE & W 9TH ST; LANSDALE; Station 345; 2015-12-10 @ 15:39:04;	Lansdale
LAUREL AVE & OAKDALE AVE; HORSHAM; Station 352; 2015-12-10 @ 16:46:48;	Horsham
COLLEGEVILLE RD & LYWISKI RD; SKIPACK; Station 336; 2015-12-10 @ 16:17:05;	Skippack
MAIN ST & OLD SUMNEYTOWN PIKE; LOWER SALFORD; Station 344; 2015-12-10 @ 16:51:42;	Lower Salford
BLUEROUTE & RAMP I476 NB TO CHEMICAL RD; PLYMOUTH; 2015-12-10 @ 17:35:41;	Plymouth
RT202 PKWY & KNAPP RD; MONTGOMERY; 2015-12-10 @ 17:33:50;	Montgomery
BROOK RD & COLWELL LN; PLYMOUTH; 2015-12-10 @ 16:32:10;	Plymouth

S3. Date-format manipulation during string extraction

Number of Examples that were required for this case: 1

INPUT	OUTPUT
MONTGOMERY AVE & WOODSIDE RD; LOWER MERION; Station 313; 2015-12-11 @ 04:11:35;	12 Nov 2015 4AM
DREYCOTT LN & W LANCASTER AVE; LOWER MERION; Station 313; 2015-12-11 @ 01:29:52;	12 Nov 2015 1AM
E LEVERING MILL RD & CONSHOHOCKEN STATE RD; LOWER MERION; 2015-12-11 @ 07:29:58;	12 Nov 2015 7AM
PENN VALLEY RD & MANOR RD; LOWER MERION; Station 313; 2015-12-10 @ 20:53:30;	12 Oct 2015 8PM
BELMONT AVE & OVERHILL RD; LOWER MERION; 2015-12- 10 @ 23:02:27;	12 Oct 2015 11PM
W MONTGOMERY AVE & PENNSWOOD RD; LOWER MERION; 2015-12-10 @ 19:25:22;	12 Oct 2015 7PM
ROSEMONT AVE & DEAD END; LOWER MERION; Station 313; 2015-12-10 @ 18:43:07;	12 Oct 2015 6PM
AVIGNON DR & DEAD END; LOWER MERION; 2015-12-10 @ 20:01:29-Station:STA24;	12 Oct 2015 8PM

S4. Concatenating strings

Number of Examples that were required for this case: 1

NOTE	
In this example, special character · represents spaces in the Output column.	

FIRST NAME	MIDDLE INITIAL	LAST NAME	OUTPUT
Laquanda		Lohmann	Laquanda·Lohmann
Claudio	A	Chew	Claudio·A·Chew
Sarah-Jane	S	Smith	Sarah-Jane·S·Smith
Brandi		Blumenthal	Brandi·Blumenthal
Jesusita	R	Journey	Jesusita·R·Journey
Hermina		Hults	Hermina·Hults
Anne-Marie	W	Jones	Anne-Marie·W·Jones
Rico		Ropp	Rico·Ropp

FIRST NAME	MIDDLE INITIAL	LAST NAME	OUTPUT
Lauren-May		Fullmer	Lauren-May·Fullmer
Marc	T	Maine	Marc·T·Maine
Angie		Adelman	Angie·Adelman
John-Paul		Smith	John-Paul·Smith
Song	W	Staller	Song·W·Staller
Jill		Jefferies	Jill·Jefferies
Ruby-Grace	M	Simmons	Ruby-Grace·M·Simmons

S5. Generating initials

Number of Examples that were required for this case: 2

FULL NAME	OUTPUT
Laquanda Lohmann	L.L.
Claudio Chew	C.C.
Sarah-Jane Smith	S.S.
Brandi Blumenthal	B.B.
Jesusita Journey	J.J.
Hermina Hults	H.H.
Anne-Marie Jones	A.J.
Rico Ropp	R.R.
Lauren-May Fullmer	L.F.
Marc Maine	M.M.
Angie Adelman	A.A.
John-Paul Smith	J.S.
Song Staller	S.S.
Jill Jefferies	J.J.
Ruby-Grace Simmons	R.S.

S6. Mapping constant values

Number of Examples that were required for this case: 3

ADMINISTRATIVE GENDER	OUTPUT
Male	0
Female	1
Unknown	2
Female	1
Female	1
Male	0
Unknown	2
Male	0
Female	1

Examples of number transformations by example

NOTE

Values in **bold** represent the examples that were provided in order to complete the transformation in the shown dataset.

N1. Rounding to nearest 10

Number of Examples that were required for this case: 1

INPUT	OUTPUT
112	110
117	120
11112	11110
11119	11120
548	550

N2. Rounding down to nearest 10

Number of Examples that were required for this case: 2

INPUT	OUTPUT
112	110
117	110

INPUT	OUTPUT
11112	11110
11119	11110
548	540

N3. Rounding to nearest 0.05

Number of Examples that were required for this case: 2

INPUT	OUTPUT
-75.5812935	-75.60
-75.2646799	-75.25
-75.3519752	-75.35
-75.343513	-75.35
-75.6033497	-75.60
-75.283245	-75.30

N4. Binning

Number of Examples that were required for this case: 1

INPUT	OUTPUT
20.16	20-25
14.32	10-15
5.44	5-10
3.84	0-5
3.73	0-5
7.36	5-10

N5. Scaling by 1000

Number of Examples that were required for this case: 1

INPUT	OUTPUT
-243	-243000
-12.5	-12500
-2345.23292	-2345232.92

INPUT	OUTPUT
-1202.3433	-1202343.3
1202.3433	1202343.3

N6. Padding

Number of Examples that were required for this case: 1

CODE	OUTPUT
5828	05828
44130	44130
49007	49007
29682	29682
4759	04759
10029	10029
7204	07204

Examples of date transformations by example

NOTE

Values in **bold** represent the examples that were provided in order to complete the transformation in the shown dataset.

D1. Extracting date parts

These Date parts were extracted using different by-example transformations on the same data set. Bold strings represent the examples that were given in their respective transformation.

DATETIME	WEEKDAY	DATE	MONTH	YEAR	HOUR	MINUTE	SECOND
31-Jan-2031 05:54:18	Fri	31	Jan	2031	5	54	18
17-Jan-1990 13:32:01	Wed	17	Jan	1990	13	32	01
14-Feb-2034 05:36:07	Tue	14	Feb	2034	5	36	07
14-Mar-2002 13:16:16	Thu	14	Mar	2002	13	16	16

DATETIME	WEEKDAY	DATE	MONTH	YEAR	HOUR	MINUTE	SECOND
21-Jan-1985 05:44:43	Mon	21	Jan	1985	5	44	43
16-Aug-1985 01:11:56	Fri	16	Aug	1985	1	11	56
20-Dec-2033 18:36:29	Tue	20	Dec	2033	18	36	29
16-Jul-1984 10:21:59	Mon	16	Jul	1984	10	21	59
13-Jan-2038 10:59:36	Wed	13	Jan	2038	10	59	36
14-Aug-1982 15:13:54	Sat	14	Aug	1982	15	13	54
22-Nov-2030 08:18:08	Fri	22	Nov	2030	8	18	08
21-Oct-1997 08:42:58	Tue	21	Oct	1997	8	42	58
28-Nov-2006 14:19:15	Tue	28	Nov	2006	14	19	15
29-Apr-2031 04:59:45	Tue	29	Apr	2031	4	59	45
29-Jan-2032 02:38:36	Thu	29	Jan	2032	2	38	36
11-May-2028 15:31:52	Thu	11	May	2028	15	31	52
15-Jul-1977 12:45:39	Fri	15	Jul	1977	12	45	39
27-Jan-2029 05:55:41	Sat	27	Jan	2029	5	55	41

DATETIME	WEEKDAY	DATE	MONTH	YEAR	HOUR	MINUTE	SECOND
03-Mar-2024 10:17:49	Sun	3	Mar	2024	10	17	49
14-Apr-2010 00:23:13	Wed	14	Apr	2010	0	23	13

D2. Formatting dates

These Date formattings were done using different by-example transformations on the same data set. Bold strings represent the examples that were given in their respective transformation.

DATETIME	FORMAT1	FORMAT2	FORMAT3	FORMAT4	FORMAT5
31-Jan-2031 05:54:18	1/31/2031	Friday, January 31, 2031	01312031 5:54	31/1/2031 5:54 AM	Q1 2031
17-Jan-1990 13:32:01	1/17/1990	Wednesday, January 17, 1990	01171990 13:32	17/1/1990 1:32 PM	Q1 1990
14-Feb-2034 05:36:07	2/14/2034	Tuesday, February 14, 2034	02142034 5:36	14/2/2034 5:36 AM	Q1 2034
14-Mar-2002 13:16:16	3/14/2002	Thursday, March 14, 2002	03142002 13:16	14/3/2002 1:16 PM	Q1 2002
21-Jan-1985 05:44:43	1/21/1985	Monday, January 21, 1985	01211985 5:44	21/1/1985 5:44 AM	Q1 1985
16-Aug-1985 01:11:56	8/16/1985	Friday, August 16, 1985	08161985 1:11	16/8/1985 1:11 AM	Q3 1985
20-Dec-2033 18:36:29	12/20/2033	Tuesday, December 20, 2033	12202033 18:36	20/12/2033 6:36 PM	Q4 2033
16-Jul-1984 10:21:59	7/16/1984	Monday, July 16, 1984	07161984 10:21	16/7/1984 10:21 AM	Q3 1984
13-Jan-2038 10:59:36	1/13/2038	Wednesday, January 13, 2038	01132038 10:59	13/1/2038 10:59 AM	Q1 2038
14-Aug-1982 15:13:54	8/14/1982	Saturday, August 14, 1982	08141982 15:13	14/8/1982 3:13 PM	Q3 1982
22-Nov-2030 08:18:08	11/22/2030	Friday, November 22, 2030	11222030 8:18	22/11/2030 8:18 AM	Q4 2030
21-Oct-1997 08:42:58	10/21/1997	Tuesday, October 21, 1997	10211997 8:42	21/10/1997 8:42 AM	Q4 1997

DATETIME	FORMAT1	FORMAT2	FORMAT3	FORMAT4	FORMAT5
28-Nov-2006 14:19:15	11/28/2006	Tuesday, November 28, 2006	11282006 14:19	28/11/2006 2:19 PM	Q4 2006
29-Apr-2031 04:59:45	4/29/2031	Tuesday, April 29, 2031	04292031 4:59	29/4/2031 4:59 AM	Q2 2031
29-Jan-2032 02:38:36	1/29/2032	Thursday, January 29, 2032	01292032 2:38	29/1/2032 2:38 AM	Q1 2032
11-May-2028 15:31:52	5/11/2028	Thursday, May 11, 2028	05112028 15:31	11/5/2028 3:31 PM	Q2 2028
15-Jul-1977 12:45:39	7/15/1977	Friday, July 15, 1977	07151977 12:45	15/7/1977 12:45 PM	Q3 1977
27-Jan-2029 05:55:41	1/27/2029	Saturday, January 27, 2029	01272029 5:55	27/1/2029 5:55 AM	Q1 2029
03-Mar-2024 10:17:49	3/3/2024	Sunday, March 3, 2024	03032024 10:17	3/3/2024 10:17 AM	Q1 2024
14-Apr-2010 00:23:13	4/14/2010	Wednesday, April 14, 2010	04142010 0:23	14/4/2010 12:23 AM	Q2 2010

D3. Mapping time to time periods

These Datetimes to period mappings were done using different by-example transformations on the same data set. Bold strings represent the examples that were given in their respective transformation.

DATETIME	PERIOD(SECONDS)	PERIOD(MINUTES)	PERIOD(TWO HOURS)	PERIOD(30 MINUTES)
31-Jan-2031 05:54:18	0-20	45-60	5AM-7AM	5:30-6:00
17-Jan-1990 13:32:01	0-20	30-45	1PM-3PM	13:30-14:00
14-Feb-2034 05:36:07	0-20	30-45	5AM-7AM	5:30-6:00
14-Mar-2002 13:16:16	0-20	15-30	1PM-3PM	13:00-13:30
21-Jan-1985 05:44:43	40-60	30-45	5AM-7AM	5:30-6:00
16-Aug-1985 01:11:56	40-60	0-15	1AM-3AM	1:00-1:30
20-Dec-2033 18:36:29	20-40	30-45	5PM-7PM	18:30-19:00
16-Jul-1984 10:21:59	40-60	15-30	9AM-11AM	10:00-10:30

DATETIME	PERIOD(SECONDS)	PERIOD(MINUTES)	PERIOD(TWO HOURS)	PERIOD(30 MINUTES)
13-Jan-2038 10:59:36	20-40	45-60	9AM-11AM	10:30-11:00
14-Aug-1982 15:13:54	40-60	0-15	3PM-5PM	15:00-15:30
22-Nov-2030 08:18:08	0-20	15-30	7AM-9AM	8:00-8:30
21-Oct-1997 08:42:58	40-60	30-45	7AM-9AM	8:30-9:00
28-Nov-2006 14:19:15	0-20	15-30	1PM-3PM	14:00-14:30
29-Apr-2031 04:59:45	40-60	45-60	3AM-5AM	4:30-5:00
29-Jan-2032 02:38:36	20-40	30-45	1AM-3AM	2:30-3:00
11-May-2028 15:31:52	40-60	30-45	3PM-5PM	15:30-16:00
15-Jul-1977 12:45:39	20-40	45-60	11AM-1PM	12:30-13:00
27-Jan-2029 05:55:41	40-60	45-60	5AM-7AM	5:30-6:00
03-Mar-2024 10:17:49	40-60	15-30	9AM-11AM	10:00-10:30
14-Apr-2010 00:23:13	0-20	15-30	11PM-1AM	0:00-0:30

Examples of composite transformations by example

TRIPDURATION	STARTTIME	START STATION ID	START STATION LATITUDE	START STATION LONGITUDE	USERTYPE	COLUMN
61	2016-01-08 16:09:32	107	42.3625	-71.08822	Subscriber	A Subscriber picked a bike from station 107, lat/long (42.363,-71.088), on Jan 08, 2016 at around 4PM. The trip duration was 61 minutes

TRIPDURATION	STARTTIME	START STATION ID	START STATION LATITUDE	START STATION LONGITUDE	USERTYPE	COLUMN
61	2016-01-17 09:28:10	74	42.373268	-71.118579	Customer	A Customer picked a bike from station 74, lat/long (42.373,-71.119), on Jan 17, 2016 at around 9AM. The trip duration was 61 minutes
62	2016-01-25 08:10:26	176	42.38674802 0450561	-71.11901879 3106079	Subscriber	A Subscriber picked a bike from station 176, lat/long (42.387,-71.119), on Jan 25, 2016 at around 8AM. The trip duration was 62 minutes
63	2016-01-08 10:10:29	107	42.3625	-71.08822	Subscriber	A Subscriber picked a bike from station 107, lat/long (42.363,-71.088), on Jan 08, 2016 at around 10AM. The trip duration was 63 minutes
64	2016-01-15 19:42:08	68	42.36507	-71.1031	Subscriber	A Subscriber picked a bike from station 68, lat/long (42.365,-71.103), on Jan 15, 2016 at around 7PM. The trip duration was 64 minutes

TRIPDURATION	STARTTIME	START STATION ID	START STATION LATITUDE	START STATION LONGITUDE	USERTYPE	COLUMN
64	2016-01-22 18:16:13	115	42.387995	-71.119084	Subscriber	A Subscriber picked a bike from station 115, lat/long (42.388,-71.119), on Jan 22, 2016 at around 6PM. The trip duration was 64 minutes
68	2016-01-18 09:51:52	178	42.35957320 1090441	-71.10129475 5935669	Subscriber	A Subscriber picked a bike from station 178, lat/long (42.360,-71.101), on Jan 18, 2016 at around 9AM. The trip duration was 68 minutes
69	2016-01-14 08:57:55	176	42.38674802 0450561	-71.11901879 3106079	Subscriber	A Subscriber picked a bike from station 176, lat/long (42.387,-71.119), on Jan 14, 2016 at around 8AM. The trip duration was 69 minutes
69	2016-01-13 22:12:55	141	42.36356015 8429884	-71.08216792 345047	Subscriber	A Subscriber picked a bike from station 141, lat/long (42.364,-71.082), on Jan 13, 2016 at around 10PM. The trip duration was 69 minutes

TRIPDURATION	STARTTIME	START STATION ID	START STATION LATITUDE	START STATION LONGITUDE	USERTYPE	COLUMN
69	2016-01-15 08:13:09	176	42.38674802 0450561	- 71.11901879 3106079	Subscriber	A Subscriber picked a bike from station 176, lat/long (42.387,-71.119), on Jan 15, 2016 at around 8AM. The trip duration was 69 minutes

Technical notes

Conditional transformations

In some cases, a single transformation cannot be found that satisfies the given examples. In such cases, Derive Column by Example Transform attempts to group the inputs based on some pattern and learn separate transformation for each group. We call this **Conditional Transformation**. **Conditional Transformation** is attempted only for transformations with a single input column.

Reference

More information about the String Transformation by Example technology can be found in [this publication](#).

Split column by example transformation

9/24/2018 • 8 minutes to read • [Edit Online](#)

NOTE

This article is deprecated. Support for earlier versions of this service will end incrementally. [View the support timeline](#). Start using the latest version with this [quickstart](#).

This transform predictively splits the content of a column on meaningful boundaries without requiring user input. The Split algorithm selects the boundaries after analyzing the content of the column. These boundaries could be defined by

- A fixed delimiter,
- Multiple, arbitrary delimiters appearing in particular contexts, or,
- Data patterns or certain entity types

Users can also control the splitting behavior in the Advanced mode where they can specify the delimiters or by provide examples of desired splitting.

In theory, Split operations can also be performed in the Workbench using a series of *Derive Column by Example* transforms. However, if there are several columns, deriving each of those individually even using by-example approach can be very time consuming. Predictive split enables easy splitting without the user needing to provide examples for each of the columns.

How to perform this transformation

1. Select the column that you want to split.
2. Select **Split Column by Example** from the **Transforms** menu. Or, Right-click on the header of the selected column and select **Split Column by Example** from the context menu. The Transform Editor opens and new columns are added next to the selected column. At this point, the Workbench analyzes the input column, determines split boundaries, and synthesizes a program to split the column as displayed in the grid. The synthesized program is executed against all the rows in the column. Delimiters, if any, are excluded from the final result.
3. You can click on the **Advanced mode** for finer control over the split transformation.
4. Review the output and Click **OK** to accept the transform.

The transform aims to produce the same number of resultant columns for all the rows. If any row cannot be split on the determined boundaries, it produces *null* for all the columns by default. This behavior can be changed in the **Advanced Mode**.

Transform editor: advanced mode

Advanced Mode provides a richer experience for Splitting columns.

Selecting **Keep Delimiter Columns** includes the Delimiters in the final result. Delimiters are excluded by default.

Specifying **Delimiters** overrides the automatic delimiter selection logic. Multiple delimiters, one in each line, can be specified as **Delimiters**. All those characters are used as delimiters to split the column.

Sometimes, splitting a value on determined boundaries produces different number of columns than the majority of others. In those cases, **Fill Direction** is used to decide the order in which the columns should be filled.

Clicking on **Show suggested examples** displays the representative rows for which user should provide an example of split. User can click on the **Up** arrow to the right of the suggested row to promote the row as an example.

User can **Delete Column** or **Insert new Columns** by Right-clicking on the header of the **Examples Table**.

User can Copy and Paste values from one Cell to another in order to provide an example of split.

User can switch between the **Basic Mode** and the **Advanced Mode** by clicking the links in the Transform Editor.

Transform editor: Send Feedback

Clicking on the **Send feedback** link opens the **Feedback** dialog with the comments box prepopulated with the parameter selections and the examples user has provided. User should review the content of the comments box and provide more details to help us understand the issue. If the user does not want to share data with Microsoft, user should delete the prepopulated example data before clicking the **Send Feedback** button.

Editing an existing transformation

A user can edit an existing **Split Column By Example** transform by selecting **Edit** option of the Transformation Step. Clicking on **Edit** opens the Transform Editor in **Advanced Mode**, and all the examples that were provided during creation of the transform are shown.

Examples of splitting on a fixed, single-character delimiter

It is common for data fields to be separated by a single fixed delimiter such as a comma in a CSV format. The Split transform attempts to infer these delimiters automatically. For example, in the following scenario it automatically infers the ":" as a delimiter.

Splitting IP addresses

The values in the first column are predictively split into four columns.

IP	IP_1	IP_2	IP_3	IP_4
192.168.0.102	192	168	0	102
192.138.0.101	192	138	0	101
192.168.0.102	192	168	0	102
192.158.1.202	192	158	1	202
192.168.0.102	192	168	0	102
192.169.1.102	192	169	1	102

Examples of splitting on multiple delimiters within particular contexts

The user's data may include many different delimiters separating different fields. Moreover, only some occurrences of a delimiting string may be a delimiter but not all. For example, in the following case the set of delimiters required is "-", "," and ":" to produce the desired output. However, not every occurrence of the ":" should be a split point, since we do not want to split the time but keep it in a single column. The Split transform infers delimiters within the contexts in which they occur in the input data rather than any possible occurrence of the delimiter. The transform is also aware of common data types such as dates and times.

Splitting store opening timings

The values in the following *Timings* column get predictively split into nine columns shown in the table under it.

TIMINGS

Monday - Friday: 7:00 am - 6:00 pm, Saturday: 9:00 am - 5:00 pm, Sunday: Closed

Monday - Friday: 9:00 am - 6:00 pm, Saturday: 4:00 am - 4:00 pm, Sunday: Closed

Monday - Friday: 8:30 am - 7:00 pm, Saturday: 3:00 am - 2:30 pm, Sunday: Closed

Monday - Friday: 8:00 am - 6:00 pm, Saturday: 2:00 am - 3:00 pm, Sunday: Closed

Monday - Friday: 4:00 am - 7:00 pm, Saturday: 9:00 am - 4:00 pm, Sunday: Closed

Monday - Friday: 8:30 am - 4:30 pm, Saturday: 9:00 am - 5:00 pm, Sunday: Closed

Monday - Friday: 5:30 am - 6:30 pm, Saturday: 5:00 am - 4:00 pm, Sunday: Closed

Monday - Friday: 8:30 am - 8:30 pm, Saturday: 6:00 am - 5:00 pm, Sunday: Closed

Monday - Friday: 8:00 am - 9:00 pm, Saturday: 9:00 am - 8:00 pm, Sunday: Closed

Monday - Friday: 10:00 am - 9:30 pm, Saturday: 9:30 am - 3:00 pm, Sunday: Closed

TIMINGS_1	TIMINGS_2	TIMINGS_3	TIMINGS_4	TIMINGS_5	TIMINGS_6	TIMINGS_7	TIMINGS_8	TIMINGS_9
Monday	Friday	7:00 am	6:00 pm	Saturday	9:00 am	5:00 pm	Sunday	Closed
Monday	Friday	9:00 am	6:00 pm	Saturday	4:00 am	4:00 pm	Sunday	Closed
Monday	Friday	8:30 am	7:00 pm	Saturday	3:00 am	2:30 pm	Sunday	Closed
Monday	Friday	8:00 am	6:00 pm	Saturday	2:00 am	3:00 pm	Sunday	Closed
Monday	Friday	4:00 am	7:00 pm	Saturday	9:00 am	4:00 pm	Sunday	Closed
Monday	Friday	8:30 am	4:30 pm	Saturday	9:00 am	5:00 pm	Sunday	Closed
Monday	Friday	5:30 am	6:30 pm	Saturday	5:00 am	4:00 pm	Sunday	Closed
Monday	Friday	8:30 am	8:30 pm	Saturday	6:00 am	5:00 pm	Sunday	Closed
Monday	Friday	8:00 am	9:00 pm	Saturday	9:00 am	8:00 pm	Sunday	Closed
Monday	Friday	10:00 am	9:30 pm	Saturday	9:30 am	3:00 pm	Sunday	Closed

Splitting IIS log

Here is another example of multiple arbitrary delimiters. This example also includes a contextual delimiter "/", which must not be split inside the URLs or file paths. It is tedious to perform this splitting using many *Derive Column by Example* transforms and giving examples for each field. Using the Split transform we can perform the predictive splitting without giving any examples.

LOGTEXT

192.128.138.20 - - [16/Oct/2016 16:22:33 -0200] "GET /images/picture.gif HTTP/1.1" 234 343 www.yahoo.com
["http://www.example.com/](http://www.example.com/)" "Mozilla/4.0 (compatible; MSIE 4)" "-"

10.128.72.213 - - [17/Oct/2016 12:43:12 +0300] "GET /news/stuff.html HTTP/1.1" 200 6233 www.aol.com
["http://www.sample.com/](http://www.sample.com/)" "Mozilla/5.0 (MSIE)" "-"

192.165.71.165 - - [12/Nov/2016 14:22:44 -0500] "GET /sample.ico HTTP/1.1" 342 7342 www.facebook.com "-" "Mozilla/5.0 (Windows; U; Windows NT 5.1; rv:1.7.3)" "-"

10.166.64.165 - - [23/Nov/2016 01:52:45 -0800] "GET /style.css HTTP/1.1" 200 2552 www.google.com
["http://www.test.com/index.html"](http://www.test.com/index.html) "Mozilla/5.0 (Windows)" "-"

192.167.1.193 - - [16/Jan/2017 22:34:56 +0200] "GET /js/ads.js HTTP/1.1" 200 23462 www.microsoft.com
["http://www.illustration.com/index.html"](http://www.illustration.com/index.html) "Mozilla/5.0 (Windows)" "-"

192.147.76.193 - - [28/Jan/2017 26:36:16 +0800] "GET /search.php HTTP/1.1" 400 1777 www.bing.com "-" "Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.1)" "-"

192.166.64.165 - - [23/Mar/2017 01:55:25 -0800] "GET /style.css HTTP/1.1" 200 2552 www.google.com
["http://www.test.com/index.html"](http://www.test.com/index.html) "Mozilla/5.0 (Windows)" "-"

11.167.1.193 - - [16/Apr/2017 11:34:36 +0200] "GET /js/ads.js HTTP/1.1" 200 23462 www.microsoft.com
["http://www.illustration.com/index.html"](http://www.illustration.com/index.html) "Mozilla/5.0 (Windows)" "-"

Gets split into:

LOG TEX T_1	LOG TEX T_2	LOG TEX T_3	LOG TEX T_4	LOG TEX T_5	LOG TEX T_6	LOG TEX T_7	LOG TEX T_8	LOG TEX T_9	LOG TEX T_10	LOG TEX T_11	LOG TEX T_12	LOG TEX T_13	LOG TEX T_14	LOG TEX T_15
192 .12 8.1 38. 20	16/ Oct /20 16	16: 22: 33	- 020 0	GET	ima ges/ pict ure. gif	HTT P	1.1	234	343	ww w.ya hoo. com	http ://w ww. exa mpl e.co m/	Moz illa	4.0	com pati ble; MSI E 4
10. 128 .72. 213	17/ Oct /20 16	12: 43: 12	+03 00	GET	new s/st uff.h tml	HTT P	1.1	200	623 3	ww w.ao l.co m	http ://w ww. sam ple.c om/	Moz illa	5.0	MSI E
192 .16 5.7 1.1 65	12/ Nov /20 16	14: 22: 44	- 050 0	GET	sam ple.i co	HTT P	1.1	342	734 2	ww w.fa ceb ook. com	-	Moz illa	5.0	Win dow s; U; Win dow s NT 5.1; rv:1. 7.3

LOG TEX T_1	LOG TEX T_2	LOG TEX T_3	LOG TEX T_4	LOG TEX T_5	LOG TEX T_6	LOG TEX T_7	LOG TEX T_8	LOG TEX T_9	LOG TEX T_10	LOG TEX T_11	LOG TEX T_12	LOG TEX T_13	LOG TEX T_14	LOG TEX T_15
10. 166 .64. 165	23/ Nov /20 16	01: 52: 45	- 080 0	GET	styl e.cs s	HTT P	1.1	200	255 2	ww w.g oogl e.co m	http ://w ww.t est.c om/inde x.htm l	Moz illa	5.0	Win dow s
192 .16 7.1. 193	16/J an/ 201 7	22: 34: 56	+02 00	GET	js/a ds.js	HTT P	1.1	200	234 62	ww w.mi crosoft.c om	http ://w ww.i llust ratio n.c om/inde x.htm l	Moz illa	5.0	Win dow s
192 .14 7.7 6.1 93	28/J an/ 201 7	26: 36: 16	+08 00	GET	sear ch.p hp	HTT P	1.1	400	177 7	ww w.bi ng.c om	-	Moz illa	4.0	com pati ble; MSI E 6.0; Win dow s NT 5.1
192 .16 6.6 4.1 65	23/ Mar /20 17	01: 55: 25	- 080 0	GET	styl e.cs s	HTT P	1.1	200	255 2	ww w.g oogl e.co m	http ://w ww.t est.c om/inde x.htm l	Moz illa	5.0	Win dow s
11. 167 .1.1 93	16/ Apr /20 17	11: 34: 36	+02 00	GET	js/a ds.js	HTT P	1.1	200	234 62	ww w.mi crosoft.c om	http ://w ww.i llust ratio n.c om/inde x.htm l	Moz illa	5.0	Win dow s

Examples of splitting without delimiters

In some cases, there are no actual delimiters, and data fields may occur contiguously next to each other. In this case, the Split transform automatically detects patterns in the data to infer probably split points. For example, in the following scenario we want to separate the amount from the currency type, and Split automatically infers the

boundary between the numeric and non-numeric data as the split point.

Splitting amount with currency symbol

AMOUNT	AMOUNT_1	AMOUNT_2
\$14	\$	14
£9	£	9
\$34	\$	34
€ 18	€	18
\$42	\$	42
\$7	\$	7
£42	£	42
\$16	\$	16
€ 16	€	16
\$15	\$	15
\$16	\$	16
€ 64	€	64

In the following example, we would like to separate the weight values from the units of measure. Again the Split inference detects the meaningful boundary automatically and prefers it over other possible delimiters such as the "." character.

Splitting weights with units

WEIGHT	WEIGHT_1	WEIGHT_2
2.27KG	2.27	KG
1L	1	L
2.5KG	2.5	KG
2KG	2	KG
1.7KGA	1.7	KGA
3KG	3	KG
2KG	2	KG
125G	125	G

WEIGHT	WEIGHT_1	WEIGHT_2
500G	500	G
1.5KGA	1.5	KGA

Technical notes

The Split transform feature is based on the **Predictive Program Synthesis** technique. In this technique, data transformation programs are learned automatically based on the input data. The programs are synthesized in a domain-specific language. The DSL is based on delimiters and fields that occur in particular regular expression contexts. More information about this technology can be found in a [recent publication on this topic](#).

Expand JSON transformation

9/24/2018 • 2 minutes to read • [Edit Online](#)

NOTE

This article is deprecated. Support for earlier versions of this service will end incrementally. [View the support timeline](#). Start using the latest version with this [quickstart](#).

The **Expand JSON** transform enables users to expand an existing column that contains valid JSON text into multiple columns.

How to perform this transformation

Follow these steps to perform this transform:

1. Select the source column that contains JSON text.
2. Select **Expand JSON** from the **Transforms** menu. Or, Right click on the header of the source column and select **Expand JSON** from the context menu.
3. Click **OK**.

New columns are added next to the source column. These columns contain properties from the next level of hierarchy in the JSON text. Property Key, if present, is used to create the name of the corresponding column. Nested properties are preserved as JSON text that user can iteratively expand or discard as needed.

Examples

The source column *Customer* is expanded into two columns *Customer.Name* and *Customer.Phone*.

CUSTOMER	CUSTOMER.NAME	CUSTOMER.PHONE
{ "Name" : "Carrie Dodson", "Phone" : "123-4567-890"}	Carrie Dodson	123-4567-890
{ "Name" : "Leonard Robledo", "Phone" : "456-7890-123"}	Leonard Robledo	456-7890-123

Tutorial: Classifying Iris using the command-line interface

10/24/2018 • 9 minutes to read • [Edit Online](#)

NOTE

This article is deprecated. Support for earlier versions of this service will end incrementally. [View the support timeline](#). Start using the latest version with this [quickstart](#).

Azure Machine Learning service (preview) are an integrated, end-to-end data science and advanced analytics solution for professional data scientists to prepare data, develop experiments and deploy models at cloud scale.

In this tutorial, you learn to use the command-line interface (CLI) tools in Azure Machine Learning preview features to:

- Set up an Experimentation account and create a workspace
- Create a project
- Submit an experiment to multiple compute targets
- Promote and register a trained model
- Deploy a web service to score new data

Prerequisites

To complete this tutorial, you need:

- Access to an Azure subscription and permissions to create resources in that subscription.
If you don't have an Azure subscription, create a [free account](#) before you begin.
- Azure Machine Learning Workbench application installed as described in [Quickstart: Install and start Azure Machine Learning service](#).

IMPORTANT

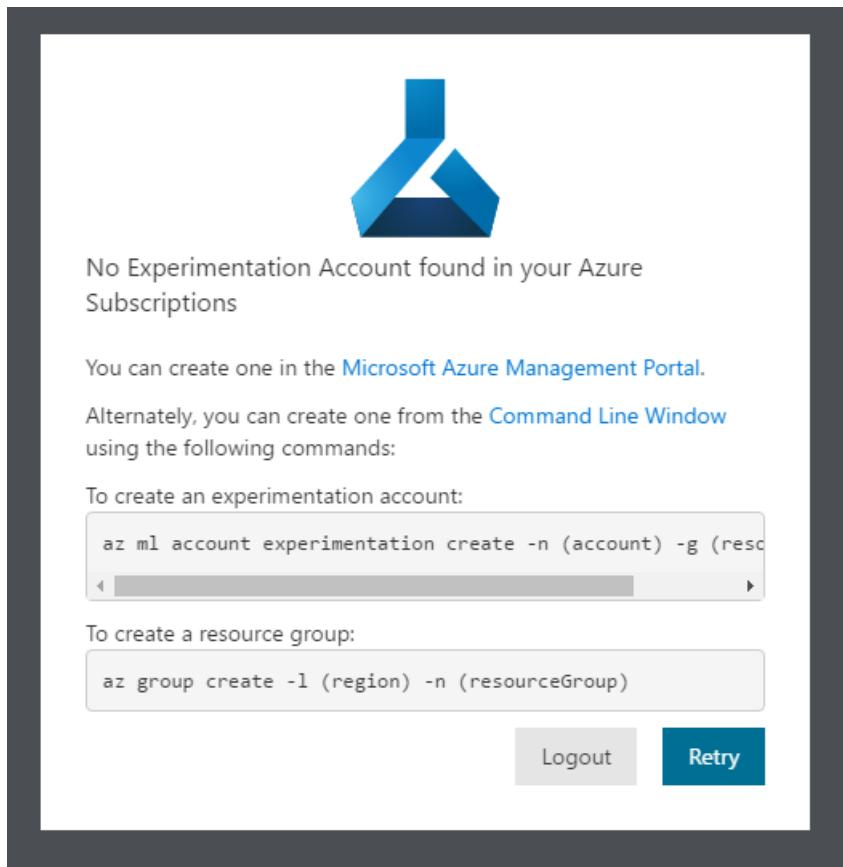
Do not create the Azure Machine Learning service accounts since you will do that using the CLI in this article.

Getting started

Azure Machine Learning command-line interface (CLI) allows you to perform all tasks required for an end-to-end data science workflow. You can access the CLI tools in the following ways:

Option 1. launch Azure ML CLI from Azure ML Workbench log-in dialog box

When you launch Azure ML Workbench and log in for the first time, and if you don't have access to an Experimentation Account already, you are presented with the following screen:



Click on the **Command Line Window** link in the dialog box to launch the command-line window.

Option 2. launch Azure ML CLI from Azure ML Workbench app

If you already have access to an Experimentation Account, you can log in successfully. And you can then open the command-line window by clicking on **File --> Open Command Prompt** menu.

Option 3. enable Azure ML CLI in an arbitrary command-line window

You can also enable Azure ML CLI in any command-line window. Do so by launching a command window, and enter the following commands:

```
# Windows Command Prompt
set PATH=%LOCALAPPDATA%\amlworkbench\Python;%LOCALAPPDATA%\amlworkbench\Python\Scripts;%PATH%

# Windows PowerShell
$env:Path =
$env:LOCALAPPDATA+"\amlworkbench\Python;"+$env:LOCALAPPDATA+"\amlworkbench\Python\Scripts;"+$env:Path

# macOS Bash Shell
PATH=$HOME/Library/Caches/AmlWorkbench/Python/bin:$PATH
```

To make the change permanent, you can use `setx` on Windows. For macOS, you can use `setenv`.

TIP

You can enable Azure CLI in your favorite terminal window by setting the preceding environment variables.

Step 1. Log in to Azure

The first step is to open the CLI from the AMLWorkbench App (File > Open Command Prompt). Doing so ensures that you have the correct python environment and that the ML CLI commands are available.

Now, you can set the right context in your CLI to access and manage Azure resources.

```
# log in  
$ az login  
  
# list all subscriptions  
$ az account list -o table  
  
# set the current subscription  
$ az account set -s <subscription id or name>
```

Step 2. Create a new Azure Machine Learning Experimentation Account and Workspace

In this step, you create a new Experimentation account and a new workspace. See [Azure Machine Learning concepts](#) for more details about experimentation accounts and workspaces.

NOTE

Experimentation accounts require a storage account, which is used to store the outputs of your experiment runs. The storage account name has to be globally unique in Azure because there is an url associated with it. If you don't specify an existing storage account, your experimentation account name is used to create a new storage account. Make sure to use a unique name, or you will get an error such as "*The storage account named <storage_account_name> is already taken.*"

Alternatively, you can use the `--storage` argument to supply an existing storage account.

```
# create a resource group  
$ az group create --name <resource group name> --location <supported Azure region>  
  
# create a new experimentation account with a new storage account  
$ az ml account experimentation create --name <experimentation account name> --resource-group <resource group name>  
  
# create a new experimentation account with an existing storage account  
$ az ml account experimentation create --name <experimentation account name> --resource-group <resource group name> --storage <storage account Azure Resource ID>  
  
# create a workspace in the experimentation account  
az ml workspace create --name <workspace name> --account <experimentation account name> --resource-group <resource group name>
```

Step 2.a (optional) Share a workspace with co-worker

Here you can explore how to share access to a workspace with a co-worker. The steps to share access to an experimentation account or to a project would be the same. Only the way of getting the Azure Resource ID would need to be updated.

```
# find the workspace Azure Resource ID  
$az ml workspace show --name <workspace name> --account <experimentation account name> --resource-group <resource group name>  
  
# add Bob to this workspace as a new owner  
$az role assignment create --assignee bob@contoso.com --role owner --scope <workspace Azure Resource ID>
```

TIP

`bob@contoso.com` in the above command must be a valid Azure AD identity in the directory where the current subscription belongs to.

Step 3. Create a new project

Our next step is to create a new project. There are several ways to get started with a new project.

Create a new blank project

```
# create a new project
$ az ml project create --name <project name> --workspace <workspace name> --account <experimentation account name> --resource-group <resource group name> --path <local folder path>
```

Create a new project with a default project template

You can create a new project with a default template.

```
$ az ml project create --name <project name> --workspace <workspace name> --account <experimentation account name> --resource-group <resource group name> --path <local folder path> --template
```

Create a new project associated with a cloud Git repository

You can create a new project associated with a Git repository in Azure Repos. Every time an experiment is submitted, a snapshot of the entire project folder is committed to the remote Git repo. See [Using Git repository with an Azure Machine Learning Workbench project](#) for more details.

NOTE

Azure Machine Learning only supports empty Git repos created in Azure Repos.

```
$ az ml project create --name <project name> --workspace <workspace name> --account <experimentation account name> --resource-group <resource group name> --path <local folder path> --repo <repo URL>
```

TIP

If you are getting an error "Repository url might be invalid or user might not have access", you can create a security token in Azure DevOps (under *Security, Add personal access tokens* menu) and use the `--vstoken` argument when creating your project.

Create a new project from a sample

In this example, you create a new project using a sample project as a template.

```
# List the project samples, find the Classifying Iris sample
$ az ml project sample list

# Create a new project from the sample
az ml project create --name <project name> --workspace <workspace name> --account <experimentation account name> --resource-group <resource group name> --path <local folder path> --template-url
https://github.com/MicrosoftDocs/MachineLearningSamples-Iris
```

Once your project is created, use `cd` command to enter the project directory.

Step 4 Run the training experiment

The following steps assume that you have a project with the Iris sample (see [Create a new project from an online sample](#)).

Prepare your environment

For the Iris sample, you must install matplotlib.

```
$ pip install matplotlib
```

Submit the experiment

```
# Execute the file  
$ az ml experiment submit --run-configuration local iris_sklearn.py
```

Iterate on your experiment with descending regularization rates

With some creativity, it's simple to put together a Python script that submits experiments with different regularization rates. (You might have to edit the file to point to the right project path.)

```
$ python run.py
```

Step 5. View run history

Following command lists all the previous runs executed.

```
$ az ml history list -o table
```

Running the preceding command displays a list of all the runs belonging to this project. You can see that accuracy and regularization rate metrics are listed too. This makes it easy to identify the best run from the list.

Step 5.a View attachment created by a given run

To view the attachment associated with a given run, you can use the info command of run history. Find a run ID of a specific run from the preceding list.

```
$ az ml history info --run <run id> --artifact driver_log
```

To download the artifacts from a run, you can use below command:

```
# Stream a given attachment  
$ az ml history info --run <run id> --artifact <artifact location>
```

Step 6. Promote artifacts of a run

One of the runs has a better AUC, so this is the one to use when creating a scoring web service to deploy to production. In order to do so, you first need to promote the artifacts into an asset.

```
$ az ml history promote --run <run id> --artifact-path outputs/model.pkl --name model.pkl
```

This creates an `assets` folder in your project directory with a `model.pkl.link` file. This link file is used to reference a promoted asset.

Step 7. Download the files to be operationalized

Download the promoted model so you can use them to create a prediction web service.

```
$ az ml asset download --link-file assets\pickle.link -d asset_download
```

Step 8. Set up your model management environment

Create an environment to deploy web services. You can run the web service on the local machine using Docker. Or deploy it to an ACS cluster for high-scale operations.

```
# Create new local operationalization environment
$ az ml env setup -l <supported Azure region> -n <env name>
# Once setup is complete, set your environment for current context
$ az ml env set -g <resource group name> -n <env name>
```

Step 9. Create a model management account

A model management account is required to deploy and track your models in production.

```
$ az ml account modelmanagement create -n <model management account name> -g <resource group name> -l
<supported Azure region>
```

Step 10. Create a web service

Create a web service that returns a prediction using the model you deployed.

```
$ az ml service create realtime -m asset_download/model.pkl -f score_iris.py -r python -n <web service name>
```

Step 11. Run the web service

Using the web service ID from the output of the previous step, call the web service and test it.

```
# Get web service usage information
$ az ml service usage realtime -i <web service id>

# Call the web service with the run command:
$ az ml service run realtime -i <web service id> -d <input data>
```

Step 12. Deleting all the resources

Let's complete this tutorial by deleting all the resources that were created, unless you want to keep working on it.

To do so, delete the resource group holding the resources.

```
az group delete --name <resource group name>
```

Next Steps

In this tutorial, you have learned to use the Azure Machine Learning to:

- Set up an experimentation account, creating workspace
- Create projects
- Submit experiments to multiple compute target
- Promote and register a trained model
- Create a model management account for model management
- Create an environment for deploying web services
- Deploy a web-service and score with new data

How to Use Run History and Model Metrics in Azure Machine Learning Workbench

9/24/2018 • 7 minutes to read • [Edit Online](#)

NOTE

This article is deprecated. Support for earlier versions of this service will end incrementally. [View the support timeline](#). Start using the latest version with this [quickstart](#).

Azure Machine Learning Workbench supports data science experimentation via its **Run History** and **Model Metrics** features. **Run History** provides a means to track the outputs of your machine learning experiments, and then enables filtering and comparison of their results. **Model Metrics** can be logged from any point of your scripts, tracking whatever values are most important in your data science experiments. This article describes how to make effective use of these features to increase the rate and the quality of your data science experimentation.

Prerequisites

To step through this how-to guide, you need to:

- [Create and Install Azure Machine Learning](#)
- [Create a Project](#)

Azure ML Logging API Overview

The [Azure ML Logging API](#) is available via the **azureml.logging** module in Python (which is installed with the Azure ML Workbench.) After importing this module, you can use the **get_azureml_logger** method to instantiate a **logger** object. Then, you can use the logger's **log** method to store key/value pairs produced by your Python scripts. Currently, logging model metrics of scalar and list types are supported as shown.

```
# create a logger instance in already set up environment
from azureml.logging import get_azureml_logger
logger = get_azureml_logger()

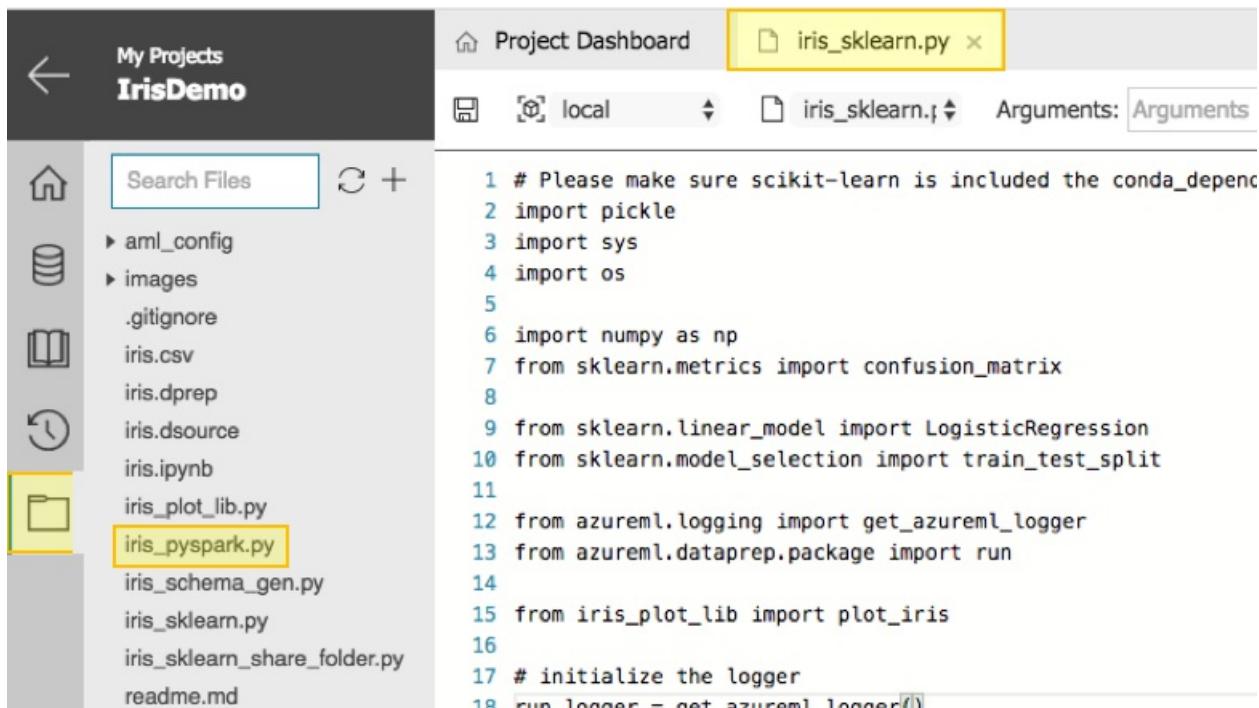
# log scalar (any integer or floating point type is fine)
logger.log("simple value", 7)

# log list
logger.log("all values", [5, 6, 7])
```

It is easy to use the logger within your Azure ML Workbench projects, and this article shows you how to do so.

Create a Project in Azure ML Workbench

If you don't already have a project, you can create one from the [Create and Install Quickstart](#) From the **Project Dashboard**, you can open the **iris_sklearn.py** script (as shown.)



The screenshot shows the Azure ML Studio interface. In the top left, there's a sidebar with icons for Home, Datasets, Models, Experiments, and Projects. The 'My Projects' section shows 'IrisDemo'. The main area is titled 'Project Dashboard' with a tab for 'iris_sklearn.py'. The file list on the left includes 'aml_config', 'images', '.gitignore', 'iris.csv', 'iris.dprep', 'iris.dsoure', 'iris.ipynb', 'iris_plot_lib.py', 'iris_pyspark.py' (which is highlighted with a yellow box), 'iris_schema_gen.py', 'iris_sklearn.py', 'iris_sklearn_share_folder.py', and 'readme.md'. The code editor on the right displays the 'iris_sklearn.py' script:

```
1 # Please make sure scikit-learn is included the conda_dependencies.txt
2 import pickle
3 import sys
4 import os
5
6 import numpy as np
7 from sklearn.metrics import confusion_matrix
8
9 from sklearn.linear_model import LogisticRegression
10 from sklearn.model_selection import train_test_split
11
12 from azureml.logging import get_azureml_logger
13 from azureml.dataprep.package import run
14
15 from iris_plot_lib import plot_iris
16
17 # initialize the logger
18 run_logger = get_azureml_logger()
```

You can use this script as a guide for expected implementation of model metric logging in Azure ML.

Parameterize and Log Model Metrics from Script

In the **iris_sklearn.py** script, the expected pattern to import and construct the logger in Python can be reduced to the following lines of code.

```
from azureml.logging import get_azureml_logger
run_logger = get_azureml_logger()
```

Once created, you can invoke the **log** method with any name/value pair.

When development is complete, it is often useful to parameterize scripts so that values can be passed in via the command line. The sample below shows how to accept command-line parameters (when present) using standard Python libraries. This script takes a single parameter for the Regularization Rate (*reg*) used to fit a classification model in an effort to increase *accuracy* without overfitting. These variables are then logged as *Regularization Rate* and *Accuracy* so that the model with optimal results can be easily identified.

```

# change regularization rate and you will likely get a different accuracy.
reg = 0.01
# load regularization rate from argument if present
if len(sys.argv) > 1:
    reg = float(sys.argv[1])

print("Regularization rate is {}".format(reg))

# log the regularization rate
run_logger.log("Regularization Rate", reg)

# train a logistic regression model on the training set
clf1 = LogisticRegression(C=1/reg).fit(X_train, Y_train)
print (clf1)

# evaluate the test set
accuracy = clf1.score(X_test, Y_test)
print ("Accuracy is {}".format(accuracy))

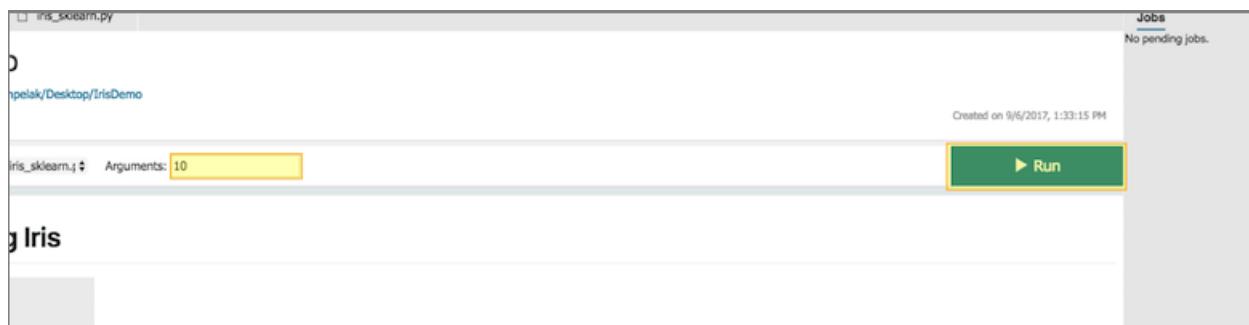
# log accuracy
run_logger.log("Accuracy", accuracy)

```

Taking these steps in your scripts enable them to make optimal usage of **Run History**.

Launch Runs from Project Dashboard

Returning to the **Project Dashboard**, you can launch a **tracked run** by selecting the **iris_sklearn.py** script and entering the **regularization rate** parameter in the **Arguments** edit box.



Since launching tracked runs does not block Azure ML Workbench, several can be launched in parallel. The status of each tracked run is visible in the **Jobs Panel** as shown.

The screenshot shows the Azure ML Workbench interface with the 'iris_sklearn.py' project selected. The top navigation bar includes a 'Run' button. On the right, a 'Jobs' sidebar lists eight completed runs of 'iris_sklearn.py' from 2017, each with a timestamp and a green checkmark. The main content area shows the script's code, which includes imports, a command-line argument, and a note about the Iris flower dataset.

This enables optimal resource utilization without requiring each job to run in serial.

View Results in Run History

Progress and results of tracked runs are available for analysis in Azure ML Workbench's **Run History**. **Run History** provides three distinct views:

- Dashboard
- Details
- Comparison

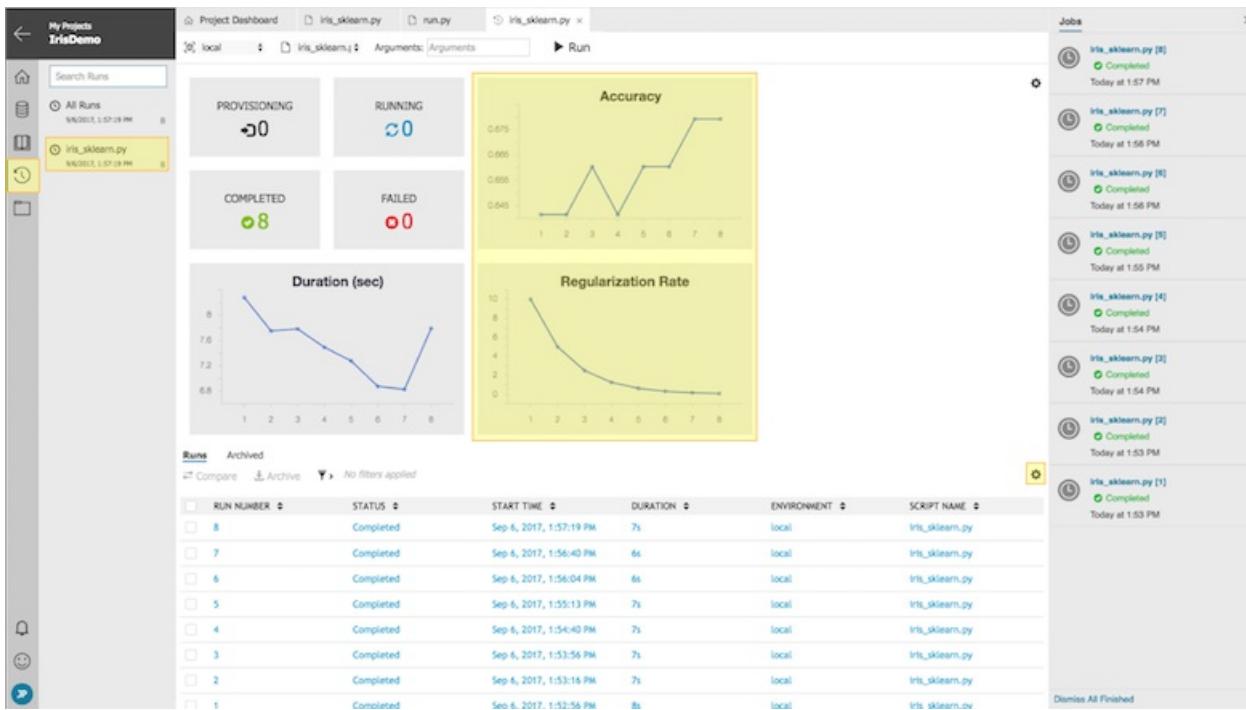
The **Dashboard** view displays data across all runs of a given script, rendered in both graphical, and tabular forms.

The **Details** view displays all data generated from a specific run of a given script, including logged metrics and output files (such as rendered plots.) The **Comparison** view enables results of two or three runs to be viewed side-by-side, also including logged metrics and output files.

Across eight tracked runs of **iris_sklearn.py**, values for the **regularization rate** parameter and **accuracy** result were logged to illustrate how to use the Run History views.

Run History Dashboard

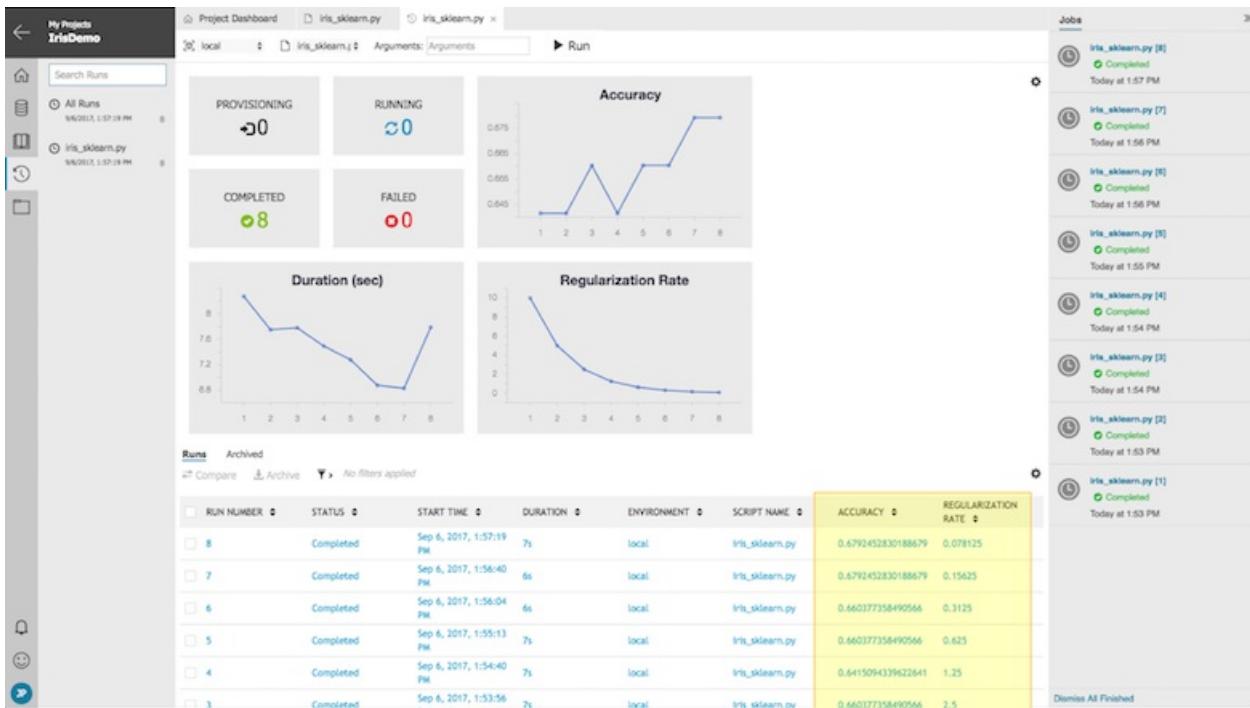
The results of all eight runs are visible in the **Run History Dashboard**. As **iris_sklearn.py** logs *Regularization Rate* and *Accuracy*, the **Run History Dashboard** displays charts for these values by default.



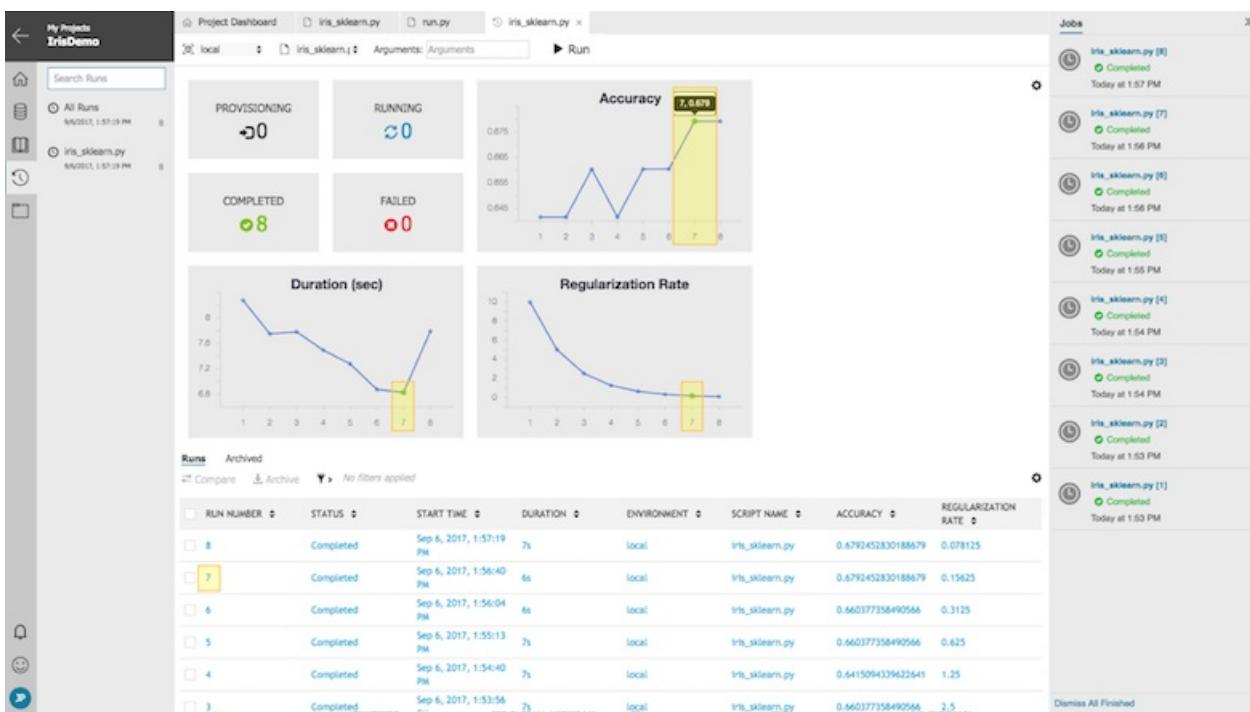
The **Run History Dashboard** can be customized so that logged values also appear in the grid. Clicking the **customize** icon displays the **List View Customization** dialogue as shown.

The screenshot shows the same Run History Dashboard as above, but with the 'customize' icon (a gear icon) selected, opening the 'List View Customization' dialog. This dialog allows users to select which columns to display in the run history grid. The 'Accuracy' and 'Regularization Rate' checkboxes are currently selected and highlighted with a yellow border. Other available columns include Run Number, Status, Start Time, Duration, Environment, Script Name, Description, Name, and Run Id. At the bottom of the dialog are 'Apply' and 'Cancel' buttons.

Any values logged during tracked runs are available for display, and selecting **Regularization Rate** and **Accuracy** adds them to the grid.



It is easy to find interesting runs by hovering over points in the charts. In this case, Run 7 yielded a good accuracy coupled with a low duration.



Clicking a point associated with Run 7 in any chart or the link to Run 7 in the grid displays the **Run History Details**.

Run History Details

From this view, full results of the Run 7 along with any artifacts produced by Run 7 are displayed.

The screenshot shows the Run History Details view for a completed run. In the Run Properties section, details like Status (Completed), Start Time (Sep 6, 2017, 1:56:40 PM), Duration (6s), and Accuracy (0.6792452830188679) are listed. The Output Files section shows files like cm.png, model.pkl, roc.png, and sdk_debug.txt. The Output Images section displays two plots: a confusion matrix heatmap and an ROC curve plot. The Log Files section shows a single file named dataprep. On the right, a list of recent runs and jobs is shown.

The **Run History Details** view also provides the capability to **download** any files written to the `./outputs` folder (these files are backed by Azure ML Workbench's cloud storage for Run History, which is the subject of another article).

Finally, **Run History Details** provides a means to restore your project its state at the time of this run. Clicking the **Restore** button displays a confirmation dialogue as shown.

The screenshot shows the Run History Details view with a modal dialog titled "Restore Project". The dialog contains a warning message: "Warning: this operation restores the current project folder to the state when this run was kicked-off. Do you want to proceed?". There is a checkbox labeled "I understand by clicking Yes, all files may be replaced or removed." and two buttons: "Yes" and "No". The background shows the Run Properties and Output Files sections of the view.

If confirmed, files may be overwritten or removed, so use this feature carefully.

Run History Comparison

Selecting two or three runs in the **Run History Dashboard** and clicking **Compare** brings you to the **Run History Comparison** view. Alternatively, clicking **Compare** and selecting a run within the **Run History Details** view also brings you to the **Run History Comparison** view. In either case, the **Run History Comparison** view provides a means to see the logged results and artifacts of two or three runs side by side.

This view is especially useful for comparison of plots, but in general, any properties of runs can be compared here.

Command Line Interface

Azure Machine Learning Workbench also provides access to Run History through its **Command Line Interface**. To access the **Command Line Interface**, click the **Open Command Prompt** menu as shown.

Run Number	Status	Start Time	Duration (sec)	Target	Script Name	Accuracy	Regularization Rate
8	Completed	Sep 7, 2017, 8:58:01 PM	6s	local	iris_sklearn.py	0.6792452830188679	0.078125

The commands available for Run History are accessed via `az ml history`, with online help available by adding the `-h` flag.

```
$ az ml history -h

Group
  az ml history: View run history of machine learning experiments.

Commands:
  compare : Compare two runs.
  download: Download all the artifacts from a run into the destination path.
  info    : Details about one run.
  last    : Get detail about most recent run.
  list    : List runs.
  promote : Promote Artifacts.
```

These commands provide the same features and return the same data shown the **Run History Views**. For example, the results of last run can be displayed as a JSON object.

```
$ az ml history last
{
  "Accuracy": 0.6792452830188679,
  "Regularization Rate": 0.078125,
  "attachments": "control_log, control_log.txt, driver_log, driver_log.txt, pid.txt,
  dataprep/backgroundProcess.log, dataprep/backgroundProcess_Engine.log,
  dataprep/backgroundProcess_EngineHost.log, dataprep/backgroundProcess_ProjectProvider.log,
  dataprep/backgroundProcess_Sampling.log, dataprep/backgroundProcess_Telemetry.log, outputs/cm.png,
  outputs/model.pkl, outputs/sdk_debug.txt, outputs/roc.png",
  "created_utc": "2017-09-08T00:58:01.611105+00:00",
  "description": null,
  "duration": "0:00:04.892389",
  "end_time_utc": "2017-09-08T00:58:07.951403+00:00",
  "experiment_id": "ce92d0a9-3e2c-4d51-85de-93ef22249cel",
  "heartbeat_enabled": true,
  "hidden": false,
  "name": null,
  "parent_run_id": null,
  "properties": {
    "CommitId": "e77a5f0df2af1a482bbe39b70bfbb16b62228cb3"
  },
  "run_id": "IrisDemo_1504832281116",
  "run_number": 8,
  "script_name": "iris_sklearn.py",
  "start_time_utc": "2017-09-08T00:58:03.059014+00:00",
  "status": "Completed",
  "target": "local",
  "user_id": "e9faf06-b0e4-4154-8374-aae34f9977b2"
}
```

Also, the list of all runs can be displayed in a tabular format.

```
$ az ml history list -o table
  Accuracy  Regularization Rate  Duration  Run_id  Script_name  Start_time_utc
  Status

  -----
  0.679245      0.078125  0:00:04.892389 IrisDemo_1504832281116 iris_sklearn.py 2017-09-
08T00:58:03.059014+00:00 Completed
  0.679245      0.15625   0:00:04.734956 IrisDemo_1504832255198 iris_sklearn.py 2017-09-
08T00:57:38.507196+00:00 Completed
  0.660377      0.3125    0:00:04.913762 IrisDemo_1504832234904 iris_sklearn.py 2017-09-
08T00:57:16.849881+00:00 Completed
  0.660377      0.625     0:00:06.107764 IrisDemo_1504832210767 iris_sklearn.py 2017-09-
08T00:56:54.602813+00:00 Completed
  0.641509      1.25      0:00:04.742727 IrisDemo_1504832171373 iris_sklearn.py 2017-09-
08T00:56:13.879280+00:00 Completed
  0.660377      2.5       0:00:04.915401 IrisDemo_1504832148526 iris_sklearn.py 2017-09-
08T00:55:52.937083+00:00 Completed
  0.641509      5         0:00:04.730627 IrisDemo_1504832127172 iris_sklearn.py 2017-09-
08T00:55:29.612382+00:00 Completed
  0.641509      10        0:00:06.059082 IrisDemo_1504832109906 iris_sklearn.py 2017-09-
08T00:55:14.739806+00:00 Completed
```

The **Command Line Interface** is an alternative pathway to access the power of Azure Machine Learning Workbench.

Next Steps

These features are available to assist with the process of data science experimentation. We hope that you find them to be useful, and would greatly appreciate your feedback. This is just our initial implementation, and we have a great deal of enhancements planned. We look forward to continuously delivering them to the Azure Machine Learning Workbench.

Find runs with the best accuracy and lowest duration

9/24/2018 • 2 minutes to read • [Edit Online](#)

Given multiple runs, one use case is to find runs with the best accuracy. One approach is to use the command-line interface (CLI) with a [JMESPath](#) query. For more information on how to use JMESPath in the Azure CLI, see [Use JMESPath queries with Azure CLI](#). In the following example, four runs are created with accuracy values of 0, 0.98, 1, and 1. Runs are filtered if they are in the range `[MaxAccuracy-Threshold, MaxAccuracy]` where `Threshold = .03`.

Sample data

If you don't have existing runs with an `Accuracy` value, the following steps generate runs for querying.

First, create a Python file in the Azure Machine Learning Workbench, name it `log_accuracy.py`, and paste in the following code:

```
from azureml.logging import get_azureml_logger

logger = get_azureml_logger()

accuracy_value = 0.5

if len(sys.argv) > 1:
    accuracy_value = float(sys.argv[1])

logger.log("Accuracy", accuracy_value)
```

Next, create a file `run.py`, and paste in the following code:

```
import os

accuracy_values = [0, 0.98, 1.0, 1.0]
for value in accuracy_values:
    os.system('az ml experiment submit -c local ./log_accuracy.py {}'.format(value))
```

Lastly, open the CLI through Workbench and run the command `python run.py` to submit four experiments. After the script finishes, you should see four more runs in the `Run History` tab.

Query the run history

The first command finds the max accuracy value.

```
az ml history list --query '@[?Accuracy != null] | max_by(@, &Accuracy).Accuracy'
```

Using this max accuracy value of `1` and a threshold value of `0.03`, the second command filters runs by using `Accuracy` and then sorts runs by `duration` ascending.

```
az ml history list --query '@[?Accuracy >= sum(`[1, -0.03]`)] | sort_by(@, &duration)'
```

NOTE

If you want a strict upper-bound check, the query format is

```
@[?Accuracy >= sum(`$max_accuracy_value, -$threshold]) && Accuracy <= `'$max_accuracy_value`]
```

If you use PowerShell, the following code uses local variables to store threshold and max accuracy:

```
$threshold = 0.03
$max_accuracy_value = az ml history list --query '[?Accuracy != null] | max_by(@, &Accuracy).Accuracy'
$find_runs_query = '[?Accuracy >= sum(`{0}, -{1}`)] | sort_by(@, &duration)' -f $max_accuracy_value,
$threshold
az ml history list --query $find_runs_query
```

Next steps

For more information on logging, see [How to use run history and model metrics in Azure Machine Learning Workbench](#).

Persisting changes and working with large files

9/24/2018 • 8 minutes to read • [Edit Online](#)

NOTE

This article is deprecated. Support for earlier versions of this service will end incrementally. [View the support timeline](#). Start using the latest version with this [quickstart](#).

With the Azure Machine Learning Experimentation service, you can configure a variety of execution targets. Some targets are local, such as a local computer or a Docker container on a local computer. Others are remote, such as a Docker container on a remote machine or an HDInsight cluster. For more information, see [Overview of Azure Machine Learning experiment execution service](#).

Before you can execute on a target, you must copy the project folder to the compute target. You must do so even with a local execution that uses a local temp folder for this purpose.

Execution isolation, portability, and reproducibility

The purpose of this design is to ensure the isolation, reproducibility, and portability of the execution. If you execute the same script twice, on the same or another compute target, you receive the same results. The changes made during the first execution shouldn't affect those in the second execution. With this design, you can treat compute targets as stateless computation resources, each having no affinity to the jobs that are executed after they are finished.

Challenges

Even though this design provides the benefits of portability and repeatability, it also brings some unique challenges.

Persisting state changes

If your script modifies the state of the compute context, the changes are not persisted for your next execution, and they're not propagated back to the client machine automatically.

More specifically, if your script creates a subfolder or writes a file, that folder or file will not be present in your project directory after execution. The files are stored in a temp folder in the compute target environment. You might use them for debugging purposes, but you cannot rely on their permanent existence.

Working with large files in the project folder

If your project folder contains any large files, you incur latency when the folder is copied to the target compute environment at the beginning of an execution. Even if the execution happens locally, there is still unnecessary disk traffic to avoid. For this reason, we currently cap the maximum project size at 25 MB.

Option 1: Use the *outputs* folder

This option is preferable if all the following conditions apply:

- Your script produces files.
- You expect the files to change with every experiment.
- You want to keep a history of these files.

The common use cases are:

- Training a model
- Creating a dataset
- Plotting a graph as an image file as part of your model-training execution

NOTE

Max size of tracked file in outputs folder after a run is 512 MB. This means if your script produces a file larger than 512 MB in the outputs folder, it is not collected there.

Additionally, you want to compare the outputs across runs, select an output file (such as a model) that was produced by a previous run, and then use it for a subsequent task (such as scoring).

You can write files to a folder named *outputs* that's relative to the root directory. The folder receives special treatment by the Experimentation service. Anything your script creates in the folder during the execution, such as a model file, data file, or plotted image file (collectively known as *artifacts*), is copied to the Azure Blob storage account that's associated with your experimentation account after the run is finished. The files become part of your run history record.

Here is a short example of code for storing a model in the *outputs* folder:

```
import os
import pickle

# m is a scikit-learn model.
# we serialize it into a mode.pkl file under the ./outputs folder.
with open(os.path.join('.', 'outputs', 'model.pkl'), 'wb') as f:
    pickle.dump(m, f)
```

You can download any artifact by browsing to the **Output Files** section of the run detail page of a particular run in Azure Machine Learning Workbench. Simply select the run, and then select the **Download** button.

Alternatively, you can enter the `az ml asset download` command in the command-line interface (CLI) window.

For a more complete example, see the `iris_sklearn.py` Python script in the *Classifying Iris* sample project.

Option 2: Use the shared folder

If you don't need to keep a history of each run's files, using a shared folder that can be accessed across independent runs can be a great option for the following scenarios:

- Your script needs to load data from local files, such as .csv files or a directory of text or image files, as training or test data.
- Your script processes raw data and writes out intermediate results, such as featurized training data from text or image files, which are used in a subsequent training run.
- Your script spits out a model, and your subsequent scoring script must pick up the model and use it for evaluation.

It is important to note that the shared folder lives locally on your chosen compute target. Therefore, this option works only if all your script runs that reference this shared folder are executed on the same compute target, and the compute target is not recycled between runs.

By taking advantage of the shared-folder feature, you can read from or write to a special folder that's identified by an environment variable, `AZUREML_NATIVE_SHARE_DIRECTORY`.

Example

Here is some sample Python code for using this share folder to read and write to a text file:

```
import os

# write to the shared folder
with open(os.environ['AZUREML_NATIVE_SHARE_DIRECTORY'] + 'test.txt', "w") as f1:
    f1.write("Hello World")

# read from the shared folder
with open(os.environ['AZUREML_NATIVE_SHARE_DIRECTORY'] + 'test.txt', "r") as f2:
    text = f2.read()
```

For a more complete example, see the *iris_sklearn_shared_folder.py* file in the *Classifying Iris* sample project.

Before you can use this feature, you have to set in the *.compute* file some simple configurations that represent the targeted execution context in the *aml_config* folder. The actual path to the folder can vary depending on the compute target you choose and the value you configure.

Configure local compute context

To enable this feature in a local compute context, simply add to your *.compute* file the following line, which represents the *local* environment (usually named *local.compute*).

```
# local.runconfig
...
nativeSharedDirectory: ~/.azureml/share
...
```

The path *~/.azureml/share* is the default base folder path. You can change it to any local absolute path that's accessible by the script run. The experimentation account name, workspace name, and project name are automatically appended to the base directory name, which becomes the full path of the shared directory. For example, the files can be written to (and retrieved from) the following path if you use the preceding default value:

```
# on Windows
C:\users\<username>\.azureml\share\<exp_acct_name>\<workspace_name>\<proj_name>\

# on macOS
/Users/<username>/\.azureml/share/<exp_acct_name>/<workspace_name>/<proj_name>/
```

Configure the Docker compute context (local or remote)

To enable this feature on a Docker compute context, you must add the following two lines to your local or remote Docker *.compute* file.

```
# docker.compute
...
sharedVolumes: true
nativeSharedDirectory: ~/.azureml/share
...
```

IMPORTANT

The **sharedVolumes** property must be set to *true* when you use the `AZUREML_NATIVE_SHARE_DIRECTORY` environment variable to access the shared folder. Otherwise, the execution fails.

The code running in the Docker container always sees this shared folder as */azureml-share/*. The folder path, as seen by the Docker container, is not configurable. Do not use this folder name in your code. Instead, always use

the environment variable name `AZUREML_NATIVE_SHARE_DIRECTORY` to refer to this folder. It is mapped to a local folder on the Docker host machine or compute context. The base directory of this local folder is the configurable value of the `nativeSharedDirectory` setting in the `.compute` file. The local path of the shared folder on the host machine, if you use the default value, is the following:

```
# Windows  
C:\users\<username>\.azureml\share\<exp_acct_name>\<workspace_name>\<proj_name>\  
  
# macOS  
/Users/<username>/\.azureml/share/<exp_acct_name>/<workspace_name>/<proj_name>/  
  
# Ubuntu Linux  
/home/<username>/\.azureml/share/<exp_acct_name>/<workspace_name>/<proj_name>/
```

NOTE

The path of the shared folder on the local disk is the same whether it's a local compute context or a local Docker compute context. This means you can even share files between a local run and a local Docker run.

You can place input data directly in these folders and expect that your local or Docker runs on the machine can pick it up. You can also write files to this folder from your local or Docker runs, and expect files get persisted in that folder, surviving the execution lifecycle.

For more information, see [Azure Machine Learning Workbench execution configuration files](#).

NOTE

The `AZUREML_NATIVE_SHARE_DIRECTORY` environment variable is not supported in an HDInsight compute context. However, it is easy to achieve the same result by explicitly using an absolute Azure Blob storage path to read from and write to the attached blob storage.

Option 3: Use external durable storage

You can use external durable storage to persist state during execution. This option is useful in the following scenarios:

- Your input data is already stored in durable storage that's accessible from the target compute environment.
- The files don't need to be part of the run history records.
- The files must be shared by executions across various compute environments.
- The files must be able to survive the compute context itself.

One such approach is to use Azure Blob storage from your Python or PySpark code. Here is a short example:

```

from azure.storage.blob import BlockBlobService
from azure.storage.blob.models import PublicAccess
import glob
import os

ACCOUNT_NAME = "<your blob storage account name>"
ACCOUNT_KEY = "<account key>"
CONTAINER_NAME = "<container name>"

blob_service = BlockBlobService(account_name=ACCOUNT_NAME, account_key=ACCOUNT_KEY)

## Create a new container if necessary, or use an existing one
blob_service.create_container(CONTAINER_NAME, fail_on_exist=False, public_access=PublicAccess.Container)

# df is a pandas DataFrame
df.to_csv('mydata.csv', sep='\t', index=False)

# Export the mydata.csv file to Blob storage.
for name in glob.iglob('mydata.csv'):
    blob_service.create_blob_from_path(CONTAINER_NAME, 'single_file.csv', name)

```

Here is a short example for attaching any arbitrary Azure Blob storage to an HDI Spark runtime:

```

def attach_storage_container(spark, account, key):
    config = spark._sc._jsc.hadoopConfiguration()
    setting = "fs.azure.account.key." + account + ".blob.core.windows.net"
    if not config.get(setting):
        config.set(setting, key)

attach_storage_container(spark, "<storage account name>", "<storage key>")

```

Conclusion

Because Azure Machine Learning executes scripts by copying the entire project folder to the target compute context, take special care with large input, output, and intermediary files. For large file transactions, you can use the special outputs folder, the shared folder that's accessible through the `AZUREML_NATIVE_SHARE_DIRECTORY` environment variable, or external durable storage.

Next steps

- Review the [Azure Machine Learning Workbench execution configuration files](#) article.
- See how the [Classifying Iris](#) tutorial project uses the outputs folder to persist a trained model.

How to Use Microsoft Machine Learning Library for Apache Spark

9/24/2018 • 2 minutes to read • [Edit Online](#)

NOTE

This article is deprecated. Support for earlier versions of this service will end incrementally. [View the support timeline](#). Start using the latest version with this [quickstart](#).

Introduction

Microsoft Machine Learning Library for Apache Spark (MMLSpark) provides tools that let you easily create scalable machine learning models for large datasets. It includes integration of SparkML pipelines with the [Microsoft Cognitive Toolkit](#) and [OpenCV](#), enabling you to:

- Ingress and pre-process image data
- Featurize images and text using pre-trained deep learning models
- Train and score classification and regression models using implicit featurization.

Prerequisites

To step through this how-to guide, you need to:

- [Install Azure Machine Learning Workbench](#)
- [Set up Azure HDInsight Spark cluster](#)

Run Your Experiment in Docker Container

Your Azure Machine Learning Workbench is configured to use MMLSpark when you run experiments in Docker container, either locally or in remote VM. This capability allows you to easily debug and experiment with your PySpark models, before running them on scale on a cluster.

To get started using an example, create a new project, and select "MMLSpark on Adult Census" Gallery example. Select "Docker" as the compute context from the project dashboard, and click "Run." Azure Machine Learning Workbench builds the Docker container to run the PySpark program, and then executes the program.

After the run has completed, you can view the results in run history view of Azure Machine Learning Workbench.

Install MMLSpark on Azure HDInsight Spark Cluster.

To complete this and the following step, you need to first [create an Azure HDInsight Spark cluster](#).

By default, Azure Machine Learning Workbench installs MMLSpark package on your cluster when you run your experiment. You can control this behavior and install other Spark packages by editing a file named *aml_config/spark_dependencies.yml* in your project folder.

```
# Spark configuration properties.  
configuration:  
  "spark.app.name": "Azure ML Experiment"  
  "spark.yarn.maxAppAttempts": 1  
  
repositories:  
  - "https://mmlspark.azureedge.net/maven"  
packages:  
  - group: "com.microsoft.ml.spark"  
    artifact: "mmlspark_2.11"  
    version: "0.9.9"
```

You can also install MMLSpark directly on your HDInsight Spark cluster using [Script Action](#).

Set up Azure HDInsight Spark Cluster as Compute Target

Open CLI window from Azure Machine Learning Workbench by going to "File" Menu and click "Open Command Prompt"

In CLI Window, run following commands:

```
az ml computetarget attach cluster --name <myhdi> --address <myhdi-ssh.azurehdinsight.net> --username  
<sshusername> --password <sshpwd>
```

```
az ml experiment prepare -c <myhdi>
```

Now the cluster is available as compute target for the project.

Run experiment on Azure HDInsight Spark cluster.

Go back to the project dashboard of "MMLSpark on Adult Census" example. Select your cluster as the compute target, and click run.

Azure Machine Learning Workbench submits the spark job to the cluster. You can monitor the progress and view the results in run history view.

Next steps

For information about MMLSpark library, and examples, see [MMLSpark GitHub repository](#)

Apache®, Apache Spark, and Spark® are either registered trademarks or trademarks of the Apache Software Foundation in the United States and/or other countries.

Model management setup

12/11/2018 • 5 minutes to read • [Edit Online](#)

NOTE

This article is deprecated. Support for earlier versions of this service will end incrementally. [View the support timeline](#). Start using the latest version with this [quickstart](#).

This document gets you started with using Azure ML model management to deploy and manage your machine learning models as web services.

Using Azure ML model management, you can efficiently deploy and manage Machine Learning models that are built using a number of frameworks including SparkML, Keras, TensorFlow, the Microsoft Cognitive Toolkit, or Python.

By the end of this document, you should be able to have your model management environment set up and ready for deploying your machine learning models.

What you need to get started

To get the most out of this guide, you should have contributor access to an Azure subscription or a resource group that you can deploy your models to. The CLI comes pre-installed on the Azure Machine Learning Workbench and on [Azure DSVMs](#).

Using the CLI

To use the command-line interfaces (CLIs) from the Workbench, click **File -> Open Command Prompt**.

On a Data Science Virtual Machine, connect and open the command prompt. Type `az ml -h` to see the options. For more details on the commands, use the `--help` flag.

On all other systems, you would have to install the CLIs.

NOTE

In a Jupyter notebook on a Linux DSVM, you can access the Azure CLI and Azure ML CLI with the command format below.

This is for a Jupyter notebook on a Linux DSVM, specifically. These commands access the current Python kernel in the notebook (e.g. the conda `py35` environment)

```
import sys
! {sys.executable} -m azure.cli login
! {sys.executable} -m azure.cli ml -h
```

Installing (or updating) on Windows

Install Python from <https://www.python.org/>. Ensure that you have selected to install pip.

Open a command prompt using Run As Administrator and run the following commands:

```
pip install -r https://aka.ms/az-ml-o16n-cli-requirements-file
```

Installing (or updating) on Linux

Run the following command from the command line, and follow the prompts:

```
sudo -i  
pip install -r https://aka.ms/az-ml-o16n-cli-requirements-file
```

Configuring Docker on Linux

In order to configure Docker on Linux for use by non-root users, follow the instructions here: [Post-installation steps for Linux](#)

NOTE

On a Linux DSVM, you can run the script below to configure Docker correctly. **Remember to log out and log back in after running the script.**

```
sudo /opt/microsoft/azureml/initial_setup.sh
```

Deploying your model

Use the CLI to deploy models as web services. The web services can be deployed locally or to a cluster.

Start with a local deployment, validate that your model and code work, then deploy to a cluster for production scale use.

To start, you need to set up your deployment environment. The environment setup is a one time task. Once the setup is complete, you can reuse the environment for subsequent deployments. See the following section for more detail.

When completing the environment setup:

- You are prompted to sign in to Azure. To sign in, use a web browser to open the page <https://aka.ms/devicelogin> and enter the provided code to authenticate.
- During the authentication process, you are prompted for an account to authenticate with. Important: Select an account that has a valid Azure subscription and sufficient permissions to create resources in the account. When the log-in is complete, your subscription information is presented and you are prompted whether you wish to continue with the selected account.

Environment Setup

To start the setup process, you need to register a few environment providers by entering the following commands:

```
az provider register -n Microsoft.MachineLearningCompute  
az provider register -n Microsoft.ContainerRegistry  
az provider register -n Microsoft.ContainerService
```

Local deployment

To deploy and test your web service on the local machine, set up a local environment using the following command. The resource group name is optional.

```
az ml env setup -l [Azure Region, e.g. eastus2] -n [your environment name] [-g [existing resource group]]
```

NOTE

Local web service deployment requires you to install Docker on the local machine.

The local environment setup command creates the following resources in your subscription:

- A resource group (if not provided, or if the name provided does not exist)
- A storage account
- An Azure Container Registry (ACR)
- An Application insights account

After setup completes successfully, set the environment to be used using the following command:

```
az ml env set -n [environment name] -g [resource group]
```

Cluster deployment

Use Cluster deployment for high-scale production scenarios. It sets up an ACS cluster with Kubernetes as the orchestrator. The ACS cluster can be scaled out to handle larger throughput for your web service calls.

To deploy your web service to a production environment, first set up the environment using the following command:

```
az ml env setup --cluster -n [your environment name] -l [Azure region e.g. eastus2] [-g [resource group]]
```

The cluster environment setup command creates the following resources in your subscription:

- A resource group (if not provided, or if the name provided does not exist)
- A storage account
- An Azure Container Registry (ACR)
- A Kubernetes deployment on an Azure Container Service (ACS) cluster
- An Application insights account

IMPORTANT

In order to successfully create a cluster environment, you will need to have contributor access on the Azure subscription or the resource group.

The resource group, storage account, and ACR are created quickly. The ACS deployment can take up to 20 minutes.

To check the status of an ongoing cluster provisioning, use the following command:

```
az ml env show -n [environment name] -g [resource group]
```

After setup is complete, you need to set the environment to be used for this deployment. Use the following command:

```
az ml env set -n [environment name] -g [resource group]
```

NOTE

After the environment is created, for subsequent deployments, you only need to use the set command above to reuse it.

Create a Model Management Account

A model management account is required for deploying models. You need to do this once per subscription, and can reuse the same account in multiple deployments.

To create a new account, use the following command:

```
az ml account modelmanagement create -l [Azure region, e.g. eastus2] -n [your account name] -g [resource group name] --sku-instances [number of instances, e.g. 1] --sku-name [Pricing tier for example S1]
```

To use an existing account, use the following command:

```
az ml account modelmanagement set -n [your account name] -g [resource group it was created in]
```

As a result of this process, the environment is ready and the model management account has been created to provide the features needed to manage and deploy Machine Learning models (see [Azure Machine Learning Model Management](#) for an overview).

Next steps

- For instructions on how to deploy web services to run on a local machine or a cluster continue on to [Deploying a Machine Learning Model as a Web Service](#).
- Try one of the many samples in the Gallery.

Scaling the cluster to manage web service throughput

10/24/2018 • 5 minutes to read • [Edit Online](#)

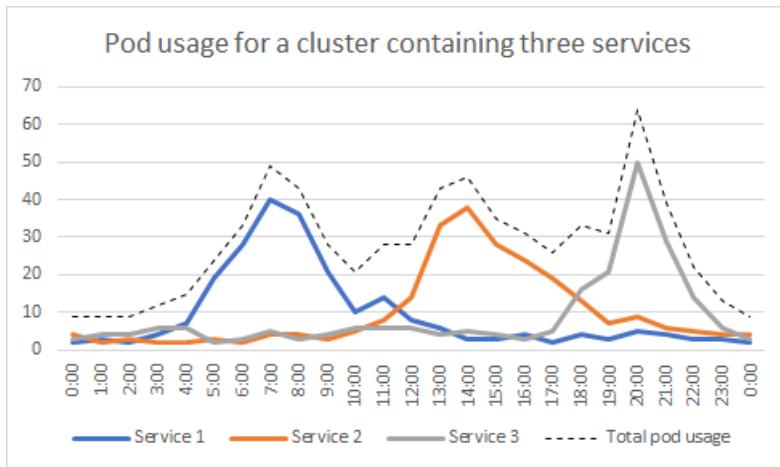
NOTE

This article is deprecated. Support for earlier versions of this service will end incrementally. [View the support timeline](#). Start using the latest version with this [quickstart](#).

Why scale the cluster?

Scaling the Azure Container Service (ACS) cluster is an effective way to optimize the service throughput while keeping the cluster size to a minimum to reduce cost.

To better understand autoscaling, consider the following example of a cluster running three web services:



The services have various peak demands: Service 1 (blue line) requires 40 pods at peak demand, Service 2 (orange line) requires 38 at peak, and Service 3 (gray line) requires 50 at peak. If you reserve the needed peak capacity for each service individually, this cluster would need at least $40 + 38 + 50 = 128$ total pods.

But consider the actual pod usage at any point in time, represented by the black dashed line in the graph. In this case, the *highest number of pods used at any one time* is 64, which occurs at 20:00 when Service 3 is at peak. At this time, Service 3 uses 50 pods, but Service 2 uses just 9 pods, and Service 1 uses only 5. Remember, this is *peak usage* for this cluster. This means that at no time does the cluster use more than 64 pods -- half the calculated requirement of 128 pods for the three services scaled independently for peak usage.

By reassigning pods in the cluster -- that is, by rescaling -- to meet the current demand of each service rather than simply require sufficient resources for the peak demand of all services, you can decrease your cluster size. In this simple example, autoscaling decreases the required number of pods from 128 to 64, cutting the required cluster size in half.

Scaling the number of pods is a relatively fast operation, requiring less than a minute, so the service's responsiveness is not seriously impacted.

NOTE

Scaling a cluster will not help with request latency issues. For operationalization purposes, scaling up should increase the number of successes and decrease Service Unavailable errors.

How to scale web services on your ACS cluster

The cluster setup option of the model management CLI by default configures two agents and one pod in your environment. It also installs the Kubernetes CLI.

You can scale the web services you have deployed to ACS by adjusting:

- The number of agent nodes in the cluster
- The number of Kubernetes pod replicas running on agent nodes

Scaling the number of nodes in the cluster

The following command sets the count of agent nodes in the cluster:

```
az acs scale -g <resource group> -n <cluster name> --new-agent-count <new scale>
```

This will take a few minutes to complete. For more information on scaling the number of nodes in the cluster, see [Scale agent nodes in a Container Service cluster](#).

Scaling the number of Kubernetes pod replicas in a cluster

You can scale the number of pod replicas assigned to the cluster using the Azure Machine Learning CLI or the [Kubernetes dashboard](#).

For more information on Kubernetes replica pods, see the [Kubernetes Pods](#) documentation.

Scaling a cluster with the Azure Machine Learning CLI

There are two ways to scale a cluster using the CLI:

- Autoscale
- Static scale

Autoscale is active by default when the service is created, and in most situations is the preferred scaling method.

Autoscale

The following command enables auto-scale, and sets the minimum and maximum number of replicas for the service.

```
az ml service update realtime -i <service id> --autoscale-enabled true --autoscale-min-replicas <positive number> --autoscale-max-replicas <positive number>
```

For example, setting `autoscale-min-replicas` to 5 will create five replicas. To arrive at an optimum number for the web service, set the number to values such as 10 and monitor the number of 503 error messages. Then adjust the number accordingly.

PARAMETER NAME	TYPE	DESCRIPTION
<code>autoscale-enabled</code>	boolean	Specifies whether Autoscale is enabled. Default: true

PARAMETER NAME	TYPE	DESCRIPTION
<code>autoscale-min-replicas</code>	integer	Specifies the minimum number of pods. Must be 0 or greater. Default: 1
<code>autoscale-max-replicas</code>	integer	Specifies the maximum number of pods. Must be 1 or greater. If <code>autoscale-max-replicas</code> is smaller than <code>autoscale-min-replicas</code> , <code>autoscale-max-replicas</code> will be ignored. Default: 10
<code>autoscale-refresh-period-seconds</code>	integer	Specifies the duration in seconds between autoscale refreshes. Default: 1
<code>autoscale-target-utilization</code>	integer	Specifies the percent utilization that autoscale targets, between 1 and 100. Default: 70

Autoscale works to ensure the following two conditions:

1. The target utilization is met
2. Scaling never exceeds the minimum and maximum settings

Services in a cluster compete for cluster resources. An autoscaled service will increase its cluster resource usage as its requests per second (RPS) increases, and will slowly release resources as the RPS decreases. Cluster resources will be acquired on demand as long as such resources exist for the service to acquire.

For more information on using the autoscale parameters, see the [Model Management Command Line Interface Reference](#) documentation.

Static scale

In general, static scaling should be avoided, since it does not allow the cluster size reduction of autoscaling. Even so, in some situations static scaling might be advised. For example, when a cluster is dedicated to a single service, autoscaling provides no benefit; all cluster resources should be assigned to that service.

In order to statically scale a cluster, autoscaling must be turned off. Disable autoscale with the following command:

```
az ml service update realtime -i <service id> --autoscale-enabled false
```

After turning off autoscale, the following command directly scales the number of replicas for a service.

```
az ml service update realtime -i <service id> -z <replica count>
```

For more information on scaling the number of nodes in the cluster, see [Scale agent nodes in a Container Service cluster](#).

Scaling number of replicas using the Kubernetes dashboard

At the command line, enter:

```
kubectl proxy
```

On Windows, the Kubernetes install location is not automatically added to the path. First navigate to the install folder:

```
c:\users\<user name>\bin
```

Once you run the command, you should see the following informational message:

```
Starting to serve on 127.0.0.1:8001
```

If the port is already in use, you see a message similar to the following example:

```
F0612 21:49:22.459111 59621 proxy.go:137] listen tcp 127.0.0.1:8001: bind: address already in use
```

You can specify an alternate port number using the `--port` parameter.

```
kubectl proxy --port=8010
Starting to serve on 127.0.0.1:8010
```

Once you have started the dashboard server, open a browser and enter the following URL:

```
127.0.0.1:<port number>/ui
```

From the dashboard main screen, click **Deployments** on the left navigation bar. If the navigation pane does not display, select this icon  on the upper left.

Locate the deployment to modify and click this icon  on the right and then click **View/Edit YAML**.

On the Edit deployment screen, locate the `spec` node, modify the `replicas` value, and click **Update**.

Enable SSL on an Azure Machine Learning Compute (MLC) cluster

9/24/2018 • 2 minutes to read • [Edit Online](#)

NOTE

This article is deprecated. Support for earlier versions of this service will end incrementally. [View the support timeline](#). Start using the latest version with this [quickstart](#).

These instructions allow you to set up SSL for scoring calls on a Machine Learning Compute (MLC) cluster.

Prerequisites

1. Decide on a CNAME (DNS name) to use when making real-time scoring calls.
2. Create a *password-free* certificate with that CNAME.
3. If the certificate is not already in PEM format, convert it to PEM using tools such as *openssl*.

You will have two files after completing the prerequisites:

- A file for the certificate, for example, `cert.pem`. Make sure the file has the full certificate chain.
- A file for the key, for example, `key.pem`

Set up an SSL certificate on a new ACS cluster

Using the Azure Machine Learning CLI, run the following command with the `-c` switch to create an ACS cluster with an SSL certificate attached:

```
az ml env create -c -g <resource group name> -n <cluster name> --cert-cname <CNAME> --cert-pem <path to cert.pem file> --key-pem <path to key.pem file>
```

Set up an SSL certificate on an existing ACS cluster

If you are targeting a cluster that was created without SSL, you can add a certificate using Azure PowerShell cmdlets.

You must provide the key and certificate in raw PEM format. These can be read into PowerShell variables:

```
$keyValueInPemFormat = [IO.File]::ReadAllText('<path to key.pem file>')
$certValueInPemFormat = [IO.File]::ReadAllText('<path to cert.pem file>')
```

Add the certificate to the cluster:

```
Set-AzureRmMlOpCluster -ResourceGroupName my-rg -Name my-cluster -SslStatus Enabled -SslCertificate $certValueInPemFormat -SslKey $keyValueInPemFormat -SslCName foo.mycompany.com
```

Map the CNAME and the IP Address

Create a mapping between the CNAME you selected in the prerequisites and the IP address of the real-time front-end (FE). To discover the IP address of the FE, run the command below. The output displays a field called "publicIpAddress", which contains the IP address of the real-time cluster front end. Refer to the instructions of your DNS provider to set up a record from the FQDN used in CNAME to the public IP address.

Auto-detection of a certificate

When you create a real-time web service using the "`az ml service create realtime`" command, Azure Machine Learning auto-detects the SSL set up on the cluster and automatically sets up the scoring URL with the designated certificate.

To see the certificate status, run the following command. Notice the "https" prefix of the scoring URI and the CNAME in the host name.

```
az ml service show realtime -n <service name>
{
    "id" : "<your service id>",
    "name" : "<your service name >",
    "state" : "Succeeded",
    "createdAt" : "<datetime>",
    "updatedAt" : "< datetime>",
    "scoringUri" : "https://<your CNAME>/api/v1/service/<service name>/score"
}
```

Deploying a Machine Learning Model as a web service

9/28/2018 • 5 minutes to read • [Edit Online](#)

NOTE

This article is deprecated. Support for earlier versions of this service will end incrementally. [View the support timeline](#). Start using the latest version with this [quickstart](#).

Azure Machine Learning Model Management provides interfaces to deploy models as containerized Docker-based web services. You can deploy models you create using frameworks such as Spark, the Microsoft Cognitive Toolkit (CNTK), Keras, Tensorflow, and Python.

This document covers the steps to deploy your models as web services using the Azure Machine Learning Model Management command-line interface (CLI).

What you need to get started

To get the most out of this guide, you should have contributor access to an Azure subscription or a resource group that you can deploy your models to. The CLI comes pre-installed on the Azure Machine Learning Workbench and on [Azure DSVMs](#). It can also be installed as a stand-alone package.

In addition, a model management account and deployment environment must already be set up. For more info on setting up your model management account and environment for local and cluster deployment, see [Model Management configuration](#).

Deploying web services

Using the CLIs, you can deploy web services to run on the local machine or on a cluster.

We recommend starting with a local deployment. You first validate that your model and code work, then deploy the web service to a cluster for production-scale use.

The following are the deployment steps:

1. Use your saved, trained, Machine Learning model
2. Create a schema for your web service's input and output data
3. Create a Docker-based container image
4. Create and deploy the web service

1. Save your model

Start with your saved trained model file, for example, mymodel.pkl. The file extension varies based on the platform you use to train and save the model.

As an example, you can use Python's Pickle library to save a trained model to a file. Here is an [example](#) using the Iris dataset:

```
import pickle
from sklearn import datasets
iris = datasets.load_iris()
X, y = iris.data, iris.target
clf = linear_model.LogisticRegression()
clf.fit(X, y)
saved_model = pickle.dumps(clf)
```

2. Create a schema.json file

While schema generation is optional, it is highly recommended to define the request and input variable format for better handling.

Create a schema to automatically validate the input and output of your web service. The CLIs also use the schema to generate a Swagger document for your web service.

To create the schema, import the following libraries:

```
from azureml.api.schema.dataTypes import DataTypes
from azureml.api.schema.sampleDefinition import SampleDefinition
from azureml.api.realtime.services import generate_schema
```

Next, define the input variables such as a Spark dataframe, Pandas dataframe, or NumPy array. The following example uses a Numpy array:

```
inputs = {"input_array": SampleDefinition(DataTypes.NUMPY, yourinputarray)}
generate_schema(run_func=run, inputs=inputs, filepath='./outputs/service_schema.json')
```

The following example uses a Spark dataframe:

```
inputs = {"input_df": SampleDefinition(DataTypes.SPARK, yourinputdataframe)}
generate_schema(run_func=run, inputs=inputs, filepath='./outputs/service_schema.json')
```

The following example uses a PANDAS dataframe:

```
inputs = {"input_df": SampleDefinition(DataTypes.PANDAS, yourinputdataframe)}
generate_schema(run_func=run, inputs=inputs, filepath='./outputs/service_schema.json')
```

The following example uses a generic JSON format:

```
inputs = {"input_json": SampleDefinition(DataTypes.STANDARD, yourinputjson)}
generate_schema(run_func=run, inputs=inputs, filepath='./outputs/service_schema.json')
```

3. Create a score.py file

You provide a score.py file, which loads your model and returns the prediction result(s) using the model.

The file must include two functions: init and run.

Add following code at the top of the score.py file to enable data collection functionality that helps collect model input and prediction data

```
from azureml.datacollector import ModelDataCollector
```

Check [model data collection](#) section for more details on how to use this feature.

Init function

Use the init function to load the saved model.

Example of a simple init function loading the model:

```
def init():
    from sklearn.externals import joblib
    global model, inputs_dc, prediction_dc
    model = joblib.load('model.pkl')

    inputs_dc = ModelDataCollector('model.pkl', identifier="inputs")
    prediction_dc = ModelDataCollector('model.pkl', identifier="prediction")
```

Run function

The run function uses the model and the input data to return a prediction.

Example of a simple run function processing the input and returning the prediction result:

```
def run(input_df):
    # clf2 is the same model as clf1, but loaded from the model.pkl file
    global clf2, inputs_dc, prediction_dc
    try:
        prediction = model.predict(input_df)
        inputs_dc.collect(input_df)
        prediction_dc.collect(prediction)
        return prediction
    except Exception as e:
        return (str(e))
```

4. Register a model

Following command is used to register a model created in step 1 above,

```
az ml model register --model [path to model file] --name [model name]
```

5. Create manifest

Following command helps create a manifest for the model,

```
az ml manifest create --manifest-name [your new manifest name] -f [path to score file] -r [runtime for the
image, e.g. spark-py]
```

You can add a previously registered model to the manifest by using argument `--model-id` or `-i` in the command shown above. Multiple models can be specified with additional `-i` arguments.

6. Create an image

You can create an image with the option of having created its manifest before.

```
az ml image create -n [image name] --manifest-id [the manifest ID]
```

NOTE

You can also use a single command to perform the model registration, manifest and model creation. Use `-h` with the service create command for more details.

As an alternative, there is a single command to register a model, create a manifest and create an image (but not

create and deploy the web service, yet) as one step as follows.

```
az ml image create -n [image name] --model-file [model file or folder path] -f [code file, e.g. the score.py file] -r [the runtime e.g. spark-py which is the Docker container image base]
```

NOTE

For more details on the command parameters, type -h at the end of the command for example, az ml image create -h.

7. Create and deploy the web service

Deploy the service using the following command:

```
az ml service create realtime --image-id <image id> -n <service name>
```

NOTE

You can also use a single command to perform all previous 4 steps. Use -h with the service create command for more details.

As an alternative, there is a single command to register a model, create a manifest, create an image, as well as, create and deploy the webservice, as one step as follows.

```
az ml service create realtime --model-file [model file/folder path] -f [scoring file e.g. score.py] -n [your service name] -s [schema file e.g. service_schema.json] -r [runtime for the Docker container e.g. spark-py or python] -c [conda dependencies file for additional python packages]
```

8. Test the service

Use the following command to get information on how to call the service:

```
az ml service usage realtime -i <service id>
```

Call the service using the run function from the command prompt:

```
az ml service run realtime -i <service id> -d "{\"input_df\": [INPUT DATA]}"
```

The following example calls a sample Iris web service:

```
az ml service run realtime -i <service id> -d "{\"input_df\": [{\"sepal length\": 3.0, \"sepal width\": 3.6, \"petal width\": 1.3, \"petal length\":0.25}]}"
```

Next steps

Now that you have tested your web service to run locally, you can deploy it to a cluster for large-scale use. For details on setting up a cluster for web service deployment, see [Model Management Configuration](#).

Customize the container image used for Azure ML Models

12/11/2018 • 2 minutes to read • [Edit Online](#)

NOTE

This article is deprecated. Support for earlier versions of this service will end incrementally. [View the support timeline](#). Start using the latest version with this [quickstart](#).

This article describes how to customize a container image for Azure Machine Learning models. Azure ML Workbench uses containers for deploying machine learning models. The models are deployed along with their dependencies, and Azure ML builds an image from the model, the dependencies, and associated files.

How to customize the Docker image

Customize the Docker image that Azure ML deploys using:

1. A `dependencies.yml` file: to manage dependencies that are installable from PyPi, you can use the `conda_dependencies.yml` file from the Workbench project, or create your own. This is the recommended approach for installing Python dependencies that are pip-installable.

Example CLI command:

```
az ml image create -n <my Image Name> --manifest-id <my Manifest ID> -c amlconfig\conda_dependencies.yml
```

Example `conda_dependencies` file:

```
name: project_environment
dependencies:
  - python=3.5.2
  - scikit-learn
  - ipykernel=4.6.1

  - pip:
    - azure-ml-api-sdk==0.1.0a11
    - matplotlib
```

2. A Docker steps file: using this option, you customize the deployed image by installing dependencies that cannot be installed from PyPi.

The file should include Docker installation steps like a DockerFile. The following commands are allowed in the file:

RUN, ENV, ARG, LABEL, EXPOSE

Example CLI command:

```
az ml image create -n <my Image Name> --manifest-id <my Manifest ID> --docker-file <myDockerStepsFileName>
```

Image, Manifest, and Service commands accept the dockerfile flag.

Example Docker steps file:

```
# Install tools required to build the project  
RUN apt-get update && apt-get install -y git libxml2-dev  
# Install library dependencies  
RUN dep ensure -vendor-only
```

NOTE

The base image for Azure ML containers is Ubuntu and can't be changed. If you specify a different base image, it will be ignored.

Next steps

Now that you've customized your container image, you can deploy it to a cluster for large-scale use. For details on setting up a cluster for web service deployment, see [Model Management Configuration](#).

Deploy an Azure Machine Learning model to an Azure IoT Edge device

11/16/2018 • 2 minutes to read • [Edit Online](#)

NOTE

This article is deprecated. Support for earlier versions of this service will end incrementally. [View the support timeline](#). Start using the latest version with this [quickstart](#).

Azure Machine Learning models can be containerized as Docker-based web services. Azure IoT Edge enables you to deploy containers remotely onto devices. Use these services together to run your models at the edge for faster response times and less data transfer.

Additional scripts and instructions can be found in the [AI Toolkit for Azure IoT Edge](#).

Operationalize the model

Azure IoT Edge modules are based on container images. To deploy your Machine Learning model to an IoT Edge device, you need to create a Docker image.

Operationalize your model by following the instructions in [Azure Machine Learning Model Management Web Service Deployment](#) to create a Docker image with your model.

Deploy to Azure IoT Edge

Once you have the image of your model, you can deploy it to any Azure IoT Edge device. All Machine Learning models can run on IoT Edge devices.

Set up an IoT Edge device

Use the Azure IoT Edge documentation to prepare a device.

1. [Register a device with Azure IoT Hub](#). The output of this processes is a connection string that you can use to configure your physical device.
2. Install the IoT Edge runtime on your physical device, and configure it with a connection string. You can install the runtime on [Windows](#) or [Linux](#) devices.

Find the Machine Learning container image location

You need the location of your Machine Learning container image. To find the container image location:

1. Log into the [Azure portal](#).
2. In the **Azure Container Registry**, select the registry you wish to inspect.
3. In the registry, click **Repositories** to see a list of all the repositories and their images.

While you're looking at your container registry in the Azure portal, retrieve the container registry credentials. These credentials need to be given to the IoT Edge device so that it can pull the image from your private registry.

1. In the container registry, click **Access keys**.
2. **Enable** the admin user, if it isn't already.
3. Save the values for **Login server**, **Username**, and **password**.

Deploy the container image to your device

With the container image and the container registry credentials, you're ready to deploy the machine learning model to your IoT Edge device.

Follow the instructions in [Deploy IoT Edge modules from the Azure portal](#) to launch your model on your IoT Edge device.

Consuming web services

12/11/2018 • 4 minutes to read • [Edit Online](#)

NOTE

This article is deprecated. Support for earlier versions of this service will end incrementally. [View the support timeline](#). Start using the latest version with this [quickstart](#).

Once you deploy a model as a realtime web service, you can send it data and get predictions from a variety of platforms and applications. The realtime web service exposes a REST API for getting predictions. You can send data to the web service in the single or multi-row format to get one or more predictions at a time.

With the [Azure Machine Learning Web service](#), an external application synchronously communicates with a predictive model by making HTTP POST call to the service URL. To make a web service call, the client application needs to specify the API key that is created when you deploy a prediction, and put the request data into the POST request body.

NOTE

Note that API keys are only available in the cluster deployment mode. Local web services do not have keys.

Service deployment options

Azure Machine Learning Web services can be deployed to the cloud-based clusters for both production and test scenarios, and to local workstations using docker engine. The functionality of the predictive model in both cases remains the same. Cluster-based deployment provides scalable and performant solution based on Azure Container Services, while the local deployment can be used for debugging.

The Azure Machine Learning CLI and API provides convenient commands for creating and managing compute environments for service deployments using the `az ml env` option.

List deployed services and images

You can list the currently deployed services and Docker images using CLI command

```
az ml service list realtime -o table
```

Note that this command always works in the context of the current compute environment. It would not show services that are deployed in an environment that is not set to be the current. To set the environment use `az ml env set`.

Get service information

After the web service has been successfully deployed, use the following command to get the service URL and other details for calling the service endpoint.

```
az ml service usage realtime -i <web service id>
```

This command prints out the service URL, required request headers, swagger URL, and sample data for calling the service if the service API schema was provided at the deployment time.

You can test the service directly from the CLI without composing an HTTP request, by entering the sample CLI

command with the input data:

```
az ml service run realtime -i <web service id> -d "Your input data"
```

Get the service API key

To get the web service key, use the following command:

```
az ml service keys realtime -i <web service id>
```

When creating HTTP request, use the key in the authorization header: "Authorization": "Bearer "

Get the service Swagger description

If the service API schema was supplied, the service endpoint would expose a Swagger document at

<http://<ip>/api/v1/service/<service name>/swagger.json>. The Swagger document can be used to automatically generate the service client and explore the expected input data and other details about the service.

Get service logs

To understand the service behavior and diagnose problems, there are several ways to retrieve the service logs:

- CLI command `az ml service logs realtime -i <service id>`. This command works in both cluster and local modes.
- If the service logging was enabled at deployment, the service logs will also be sent to AppInsight. The CLI command `az ml service usage realtime -i <service id>` shows the AppInsight URL. Note that the AppInsight logs may be delayed by 2-5 mins.
- Cluster logs can be viewed through Kubernetes console that is connected when you set the current cluster environment with `az ml env set`
- Local docker logs are available through the docker engine logs when the service is running locally.

Call the service using C#

Use the service URL to send a request from a C# Console App.

1. In Visual Studio, create a new Console App:
 - In the menu, click, File -> New -> Project
 - Under Visual Studio C#, click Windows Class Desktop, then select Console App.
2. Enter `MyFirstService` as the Name of the project, then click OK.
3. In Project References, set references to `System.Net`, and `System.Net.Http`.
4. Click Tools -> NuGet Package Manager -> Package Manager Console, then install the **Microsoft.AspNet.WebApi.Client** package.
5. Open **Program.cs** file, and replace the code with the following code:
6. Update the `SERVICE_URL` and `API_KEY` parameters with the information from your web service.
7. Run the project.

```

using System;
using System.Collections.Generic;
using System.Net.Http;
using System.Net.Http.Headers;
using Newtonsoft.Json;

namespace MyFirstService
{
    class Program
    {
        // Web Service URL (update it with your service url)
        private const string SERVICE_URL = "http://<service ip address>:80/api/v1/service/<service name>/score";
        private const string API_KEY = "your service key";

        static void Main(string[] args)
        {
            Program.PostRequest();
            Console.ReadLine();
        }

        private static void PostRequest()
        {
            HttpClient client = new HttpClient();
            client.BaseAddress = new Uri(SERVICE_URL);
            //For local web service, comment out this line.
            client.DefaultRequestHeaders.Authorization = new AuthenticationHeaderValue("Bearer", API_KEY);

            var inputJson = new List<RequestPayload>();
            RequestPayload payload = new RequestPayload();
            List<InputDf> inputDf = new List<InputDf>();
            inputDf.Add(new InputDf()
            {
                feature1 = value1,
                feature2 = value2,
            });
            payload.Input_df_list = inputDf;
            inputJson.Add(payload);

            try
            {
                var request = new HttpRequestMessage(HttpMethod.Post, string.Empty);
                request.Content = new StringContent(JsonConvert.SerializeObject(payload));
                request.Content.Headers.ContentType = new MediaTypeHeaderValue("application/json");
                var response = client.SendAsync(request).Result;

                Console.WriteLine(response.Content.ReadAsStringAsync().Result);
            }
            catch (Exception e)
            {
                Console.WriteLine(e.Message);
            }
        }

        public class InputDf
        {
            public double feature1 { get; set; }
            public double feature2 { get; set; }
        }

        public class RequestPayload
        {
            [JsonProperty("input_df")]
            public List<InputDf> Input_df_list { get; set; }
        }
    }
}

```

Call the web service using Python

Use Python to send a request to your real-time web service.

1. Copy the following code sample to a new Python file.
2. Update the data, url, and api_key parameters. For local web services, remove the 'Authorization' header.
3. Run the code.

```
import requests
import json

data = "{\"input_df\": [{\"feature1\": value1, \"feature2\": value2}]}"
body = str.encode(json.dumps(data))

url = 'http://<service ip address>:80/api/v1/service/<service name>/score'
api_key = 'your service key'
headers = {'Content-Type':'application/json', 'Authorization':('Bearer ' + api_key)}

resp = requests.post(url, body, headers=headers)
resp.text
```

Collect model data by using data collection

9/24/2018 • 3 minutes to read • [Edit Online](#)

NOTE

This article is **deprecated**. Support for earlier versions of this service will end incrementally. [View the support timeline](#). Start using the latest version with this [quickstart](#).

You can use the model data collection feature in Azure Machine Learning to archive model inputs and predictions from a web service.

Install the data collection package

You can install the data collection library natively on Linux and Windows.

Windows

On Windows, install the data collector module by using the following command:

```
pip install azureml.datacollector
```

Linux

On Linux, install the libxml++ library first. Run the following command, which must be issued under sudo:

```
sudo apt-get install libxml++2.6-2v5
```

Then run the following command:

```
pip install azureml.datacollector
```

Set environment variables

Model data collection depends on two environment variables. AML_MODEL_DC_STORAGE_ENABLED must be set to **true** (all lowercase) and AML_MODEL_DC_STORAGE must be set to the connection string for the Azure Storage account where you want to store the data.

These environment variables are already set for you when the web service is running on a cluster in Azure. When running locally you have to set them yourself. If you are using Docker, use the -e parameter of the docker run command to pass environment variables.

Collect data

To use model data collection, make the following changes to your scoring file:

1. Add the following code at the top of the file:

```
from azureml.datacollector import ModelDataCollector
```

2. Add the following lines of code to the `init()` function:

```
global inputs_dc, prediction_dc
inputs_dc = ModelDataCollector('model.pkl', identifier="inputs")
prediction_dc = ModelDataCollector('model.pkl', identifier="prediction")
```

3. Add the following lines of code to the `run(input_df)` function:

```
global inputs_dc, prediction_dc
inputs_dc.collect(input_df)
prediction_dc.collect(pred)
```

Make sure that the variables `input_df` and `pred` (prediction value from `model.predict()`) are initialized before you call the `collect()` function on them.

4. Use the `az ml service create realtime` command with the `--collect-model-data true` switch to create a real-time web service. This step makes sure that the model data is collected when the service is run.

```
c:\Temp\myIris> az ml service create realtime -f iris_score.py --model-file model.pkl -s service_schema.json -n irisapp -r python --collect-model-data true
```

5. To test the data collection, run the `az ml service run realtime` command:

```
C:\Temp\myIris> az ml service run realtime -i irisapp -d "ADD YOUR INPUT DATA HERE!!"
```

View the collected data

To view the collected data in blob storage:

1. Sign in to the [Azure portal](#).
2. Select **All Services**.
3. In the search box, type **Storage accounts** and select the Enter key.
4. From the **Storage accounts** search blade, select the **Storage account** resource. To determine your storage account, use the following steps:
 - a. Go to Azure Machine Learning Workbench, select the project you're working on, and open a command prompt from the **File** menu.
 - b. Enter `az ml env show -v` and check the `storage_account` value. This is the name of your storage account.

5. Select **Containers** on the resource blade menu, and then the container called **modeldata**. To see data start propagating to the storage account, you might need to wait up to 10 minutes after the first web service request. Data flows into blobs with the following container path:

```
/modeldata/<subscription_id>/<resource_group_name>/<model_management_account_name>/<webservice_name>/<model_id>-<model_name>-<model_version>/<identifier>/<year>/<month>/<day>/data.csv
```

Data can be consumed from Azure blobs in a variety of ways, through both Microsoft software and open-source tools. Here are some examples:

- Azure Machine Learning Workbench: Open the .csv file in Azure Machine Learning Workbench by adding the .csv file as a data source.
- Excel: Open the daily .csv files as a spreadsheet.
- [Power BI](#): Create charts with data pulled from .csv data in blobs.
- [Spark](#): Create a data frame with a large portion of .csv data.
- [Hive](#): Load .csv data into a Hive table and perform SQL queries directly against the blob.

```
python var df =  
spark.read.format("com.databricks.spark.csv").option("inferSchema","true").option("header","true").load("wasb://modeldata@<storageaccount>.blob.core.windows.net/<subscription_id>/<resource_group_name>/<model_name>-<model_version>/<identifier>/<year>/<month>/<date>/*")
```

Troubleshooting service deployment and environment setup

9/24/2018 • 3 minutes to read • [Edit Online](#)

NOTE

This article is deprecated. Support for earlier versions of this service will end incrementally. [View the support timeline](#). Start using the latest version with this [quickstart](#).

The following information can help determine the cause of errors when setting up the model management environment.

Model management environment

Contributor permission required

You need contributor access to the subscription or the resource group to set up a cluster for deployment of your web services.

Resource availability

You need to have enough resources available in your subscription so you can provision the environment resources.

Subscription Caps

Your subscription may have a cap on billing which could prevent you from provisioning the environment resources. Remove that cap to enable provisioning.

Enable debug and verbose options

Use the `--debug` and `--verbose` flags in the setup command to show debug and trace information as the environment is being provisioned.

```
az ml env setup -l <location> -n <name> -c --debug --verbose
```

Service deployment

The following information can help determine the cause of errors during deployment or when calling the web service.

Service logs

The `logs` option of the service CLI provides log data from Docker and Kubernetes.

```
az ml service logs realtime -i <web service id>
```

For additional log settings, use the `--help` (or `-h`) option.

```
az ml service logs realtime -h
```

Debug and Verbose options

Use the `--debug` flag to show debug logs as the service is being deployed.

```
az ml service create realtime -m <modelfile>.pkl -f score.py -n <service name> -r python --debug
```

Use the `--verbose` flag to see additional details as the service is being deployed.

```
az ml service create realtime -m <modelfile>.pkl -f score.py -n <service name> -r python --verbose
```

Enable request logging in App Insights

Set the `-l` flag to true when creating a web service to enable request level logging. The request logs are written to the App Insights instance for your environment in Azure. Search for this instance using the environment name you used when using the `az ml env setup` command.

- Set `-l` to true when creating the service.
- Open App Insights in Azure portal. Use your environment name to find the App Insights instance.
- Once in App Insights, click on Search in the top menu to view the results.
- Or go to `Analytics` > `Exceptions` > `exceptions take | 10`.

Add error handling in scoring script

Use exception handling in your `scoring.py` script's `run` function to return the error message as part of your web service output.

Python example:

```
try:  
    <code to load model and score>  
except Exception as e:  
    return(str(e))
```

Other common problems

- If pip install of azure-cli-ml fails with the error `cannot find the path specified` on a Windows machine, you need to enable long path support. See <https://blogs.msdn.microsoft.com/jeremykuhne/2016/07/30/net-4-6-2-and-long-paths-on-windows-10/>.
- If the `env setup` command fails with `LocationNotAvailableForResourceType`, you are probably using the wrong location (region) for the machine learning resources. Make sure your location specified with the `-l` parameter is `eastus2`, `westcentralus`, or `australiaeast`.
- If the `env setup` command fails with `Resource quota limit exceeded`, make sure you have enough cores available in your subscription and that your resources are not being used up in other processes.
- If the `env setup` command fails with
`Invalid environment name. Name must only contain lowercase alphanumeric characters`, make sure the service name does not contain upper-case letters, symbols, or the underscore (`_`) (as in `my_environment`).
- If the `service create` command fails with
`Service Name: [service_name] is invalid. The name of a service must consist of lower case alphanumeric characters (etc.)`, make sure the service name is between 3 and 32 characters in length; starts and ends with lower-case alphanumeric characters; and does not contain upper-case letters, symbols other than hyphen (`-`) and period (`.`), or the underscore (`_`) (as in `my_webservice`).
- Retry if you get a `502 Bad Gateway` error when calling the web service. It normally means the container hasn't been deployed to the cluster yet.
- If you get `CrashLoopBackOff` error when creating a service, check your logs. It typically is the result of missing

dependencies in the **init** function.

Document Collection Analysis

9/24/2018 • 13 minutes to read • [Edit Online](#)

NOTE

This article is deprecated. Support for earlier versions of this service will end incrementally. [View the support timeline](#). Start using the latest version with this [quickstart](#).

This scenario demonstrates how to summarize and analyze a large collection of documents, including techniques such as phrase learning, topic modeling, and topic model analysis using Azure ML Workbench. Azure Machine Learning Workbench provides for easy scale up for very large document collection, and provides mechanisms to train and tune models within a variety of compute contexts, ranging from local compute to Data Science Virtual Machines to Spark Cluster. Easy development is provided through Jupyter notebooks within Azure Machine Learning Workbench.

Link to the Gallery GitHub repository

The public GitHub repository for this real world scenario contains all materials, including code samples, needed for this example:

<https://github.com/Azure/MachineLearningSamples-DocumentCollectionAnalysis>

Overview

With a large amount of data (especially unstructured text data) collected every day, a significant challenge is to organize, search, and understand vast quantities of these texts. This document collection analysis scenario demonstrates an efficient and automated end-to-end workflow for analyzing large document collection and enabling downstream NLP tasks.

The key elements delivered by this scenario are:

1. Learning salient multi-words phrase from documents.
2. Discovering underlying topics presented in the document collection.
3. Representing documents by the topical distribution.
4. Presenting methods for organizing, searching, and summarizing documents based on the topical content.

The methods presented in this scenario could enable a variety of critical industrial workloads, such as discovery of topic trends anomaly, document collection summarization, and similar document search. It can be applied to many different types of document analysis, such as government legislation, news stories, product reviews, customer feedbacks, and scientific research articles.

The machine learning techniques/algorithms used in this scenario include:

1. Text processing and cleaning
2. Phrase Learning
3. Topic modeling
4. Corpus summarization

5. Topical trends and anomaly detection

Prerequisites

The prerequisites to run this example are as follows:

- Make sure that you have properly installed [Azure Machine Learning Workbench](#) by following the [Install and create Quickstart](#).
- This example could be run on any compute context. However, it is recommended to run it on a multi-core machine with at least of 16-GB memory and 5-GB disk space.

Create a new Workbench project

Create a new project using this example as a template:

1. Open Azure Machine Learning Workbench
2. On the **Projects** page, click the + sign and select **New Project**
3. In the **Create New Project** pane, fill in the information for your new project
4. In the **Search Project Templates** search box, type "Document Collection Analysis" and select the template
5. Click **Create**

Data description

The dataset used in this scenario contains text summaries and associated meta data for each legislative action taken by the US Congress. The data is collected from [GovTrack.us](#), which tracks the activities of United States Congress and helps Americans participate in their national legislative process. The bulk data can be downloaded via [this link](#) using a manual script, which is not included in this scenario. The details of how to download the data could be found in the [GovTrack API documentation](#).

Data source

In this scenario, the raw data collected is a series of legislative actions introduced by the US Congress (proposed bills and resolutions) from 1973 to June 2017 (the 93rd to the 115th Congresses). The data collected is in JSON format and contains a rich set of information about the legislative actions. Refer to [this GitHub link](#) for detailed description of the data fields. For the demonstration purpose within this scenario, only a subset of data fields were extracted from the JSON files. A pre-compiled TSV file `CongressionalDataAll_Jun_2017.tsv` containing records of those legislative actions is provided in this scenario. The TSV file can be downloaded automatically either by the notebooks `1_Preprocess_Text.ipynb` under the notebook folder or `preprocessText.py` in the Python package.

Data structure

There are nine data fields in the data file. The data field names and the descriptions are listed as follows.

FIELD NAME	TYPE	DESCRIPTION	CONTAIN MISSING VALUE
<code>ID</code>	String	The ID of the bill/resolution. The format of this field is [bill_type][number]-[congress]. For example, "hconres1-93" means the bill type is "hconres" (stands for House Concurrent Resolution, refer to this document), the bill number is '1' and the congress number is '93'.	No

FIELD NAME	TYPE	DESCRIPTION	CONTAIN MISSING VALUE
Text	String	The content of the bill/resolution.	No
Date	String	The date the bill/resolution initially proposed. In a format of 'yyyy-mm-dd'.	No
SponsorName	String	The name of the primary sponsor that proposed the bill/resolution.	Yes
Type	String	The title type of the primary sponsor, either 'rep' (representative) or 'sen' (senator).	Yes
State	String	The state of the primary sponsor.	Yes
District	Integer	The district number of the primary sponsor if the title of the sponsor is a representative.	Yes
Party	String	The party of the primary sponsor.	Yes
Subjects	String	The subject terms added cumulatively by the Library of Congress to the bill. The terms are concatenated by commas. These terms are written by a human in the Library of Congress, and are not usually present when information on the bill is first published. They can be added at any time. Thus by the end of life of a bill, some subject may not be relevant anymore.	Yes

Scenario structure

The document collection analysis example is organized into two types of deliverables. The first type is a series of iPython Notebooks that show the step-by-step descriptions of the entire workflow. The second type is a Python package as well as some code examples of how to use that package. The Python package is generic enough to serve many use cases.

The files in this example are organized as follows.

FILE NAME	TYPE	DESCRIPTION
-----------	------	-------------

FILE NAME	TYPE	DESCRIPTION
aml_config	Folder	Azure Machine Learning Workbench configuration folder, refer to this documentation for detailed experiment execution configuration
Code	Folder	The code folder used to save the Python scripts and Python package
Data	Folder	The data folder used to save intermediate files
notebooks	Folder	The Jupyter notebooks folder
Code/documentAnalysis/__init__.py	Python file	The Python package init file
Code/documentAnalysis/configs.py	Python file	The configuration file used by the document analysis Python package, including predefined constants
Code/documentAnalysis/preprocessText.py	Python file	The Python file used to preprocess the data for downstream tasks
Code/documentAnalysis/phraseLearning.py	Python file	The Python file used to learn phrases from the data and transform the raw data
Code/documentAnalysis/topicModeling.py	Python file	The Python file used to train a Latent Dirichlet Allocation (LDA) topic model
Code/step1.py	Python file	Step 1 of document collection analysis: preprocess text
Code/step2.py	Python file	Step 2 of document collection analysis: phrase learning
Code/step3.py	Python file	Step 3 of document collection analysis: train and evaluate LDA topic model
Code/runme.py	Python file	Example of run all steps in one file
notebooks/1_Preprocess_Text.ipynb	iPython Notebook	Preprocess text and transform the raw data
notebooks/2_Phase_Learning.ipynb	iPython Notebook	Learn phrases from text data (after data transform)
notebooks/3_Topic_Model_Training.ipynb	iPython Notebook	Train LDA topic model
notebooks/4_Topic_Model_Summarization.ipynb	iPython Notebook	Summarize the contents of the document collection based on a trained LDA topic model

FILE NAME	TYPE	DESCRIPTION
notebooks/5_Topic_Model_Analysis.ipynb	iPython Notebook	Analyze the topical content of a collection of text documents and correlate topical information against other meta-data associated with the document collection
notebooks/6_Interactive_Visualization.ipynb	iPython Notebook	Interactive visualization of learned topic model
notebooks/winprocess.py	Python file	The python script for multiprocessing used by notebooks
README.md	Markdown file	The README markdown file

Data ingestion and transformation

The compiled dataset `CongressionalDataAll_Jun_2017.tsv` is saved in Blob Storage and is accessible both from within the notebooks and the Python scripts. There are two steps for data ingestion and transformation: preprocessing the text data, and phrase learning.

Preprocess text data

The preprocessing step applies natural language processing techniques to clean and prepare the raw text data. It serves as a precursor for the unsupervised phrase learning and latent topic modeling. The detailed step-by-step description can be found in the notebook `1_Preprocess_Text.ipynb`. There is also a Python script `step1.py` corresponds to this notebook.

In this step, the TSV data file is downloaded from the Azure Blob Storage and imported as a Pandas DataFrame. Each row element in the DataFrame is a single cohesive long string of text or a 'document'. Each document is then split into chunks of text that are likely to be sentences, phrases, or subphrases. The splitting is designed to prohibit the phrase learning process from using cross-sentence or cross-phrase word strings when learning phrases.

There are multiple functions defined for the preprocessing step both in the notebook and the Python package. The majority of the job can be achieved by calling those two lines of codes.

```
# Read raw data into a Pandas DataFrame
textDF = getData()

# Write data frame with preprocessed text out to TSV file
cleanedDataFrame = CleanAndSplitText(textDF, saveDF=True)
```

Phrase learning

The phrase learning step implements a basic framework to learn key phrases among a large collection of documents. It is described in the paper entitled "[Modeling Multiword Phrases with Constrained Phrases Tree for Improved Topic Modeling of Conversational Speech](#)", which was originally presented in the 2012 IEEE Workshop on Spoken Language Technology. The detailed implementation of phrase learning step is shown in the iPython Notebook `2_Phrase_Learning.ipynb` and the Python script `step2.py`.

This step takes the cleaned text as input and learns the most salient phrases present in a large collection of documents. The phrase learning is an iterative process that can be divided into three tasks: count n-grams, rank potential phrases by the Weighted Pointwise Mutual Information of their constituent words, and rewrite phrase to text. Those three tasks are executed sequentially in multiple iterations until the specified phrases have been learned.

In the document analysis Python package, a Python Class `PhraseLearner` is defined in the `phraseLearning.py` file. Below is the code snippet used to learn phrases.

```

# Instantiate a PhraseLearner and run a configuration
phraseLearner = PhraseLearner(cleanedDataFrame, "CleanedText", numPhrase,
                             maxPhrasePerIter, maxPhraseLength, minInstanceCount)

# The chunks of text in a list
textData = list(phraseLearner.textFrame['LowercaseText'])

# Learn most salient phrases present in a large collection of documents
phraseLearner.RunConfiguration(textData,
                                phraseLearner.learnedPhrases,
                                addSpace=True,
                                writeFile=True,
                                num_workers=cpu_count()-1)

```

NOTE

The phrase learning step is implemented with multiprocessing. However, more CPU cores do **NOT** mean a faster execution time. In our tests, the performance is not improved with more than eight cores due to the overhead of multiprocessing. It took about two and a half hours to learn 25,000 phrases on a machine with eight cores (3.6 GHz).

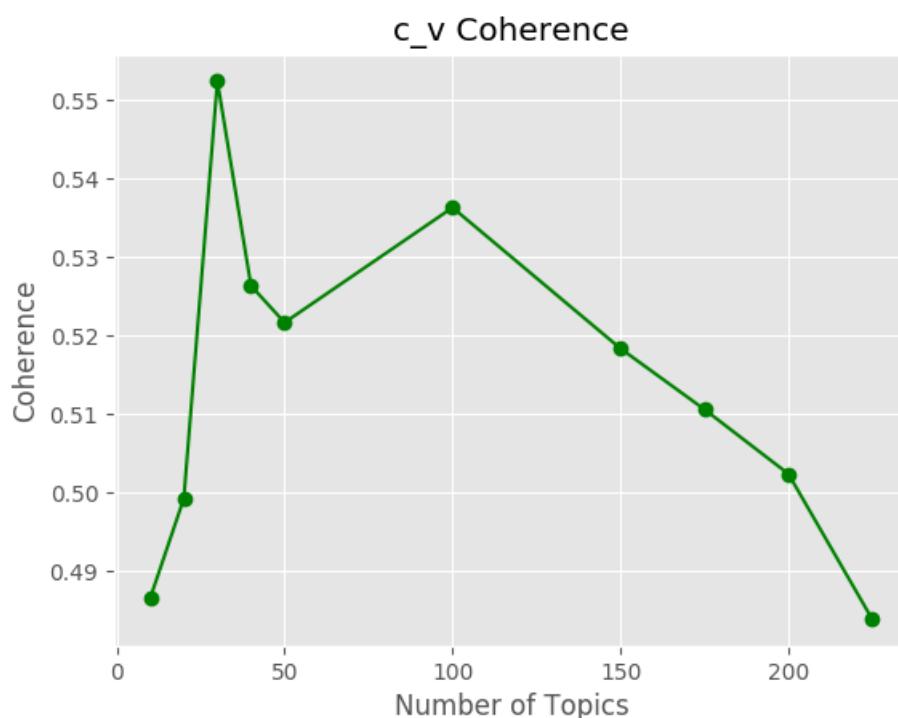
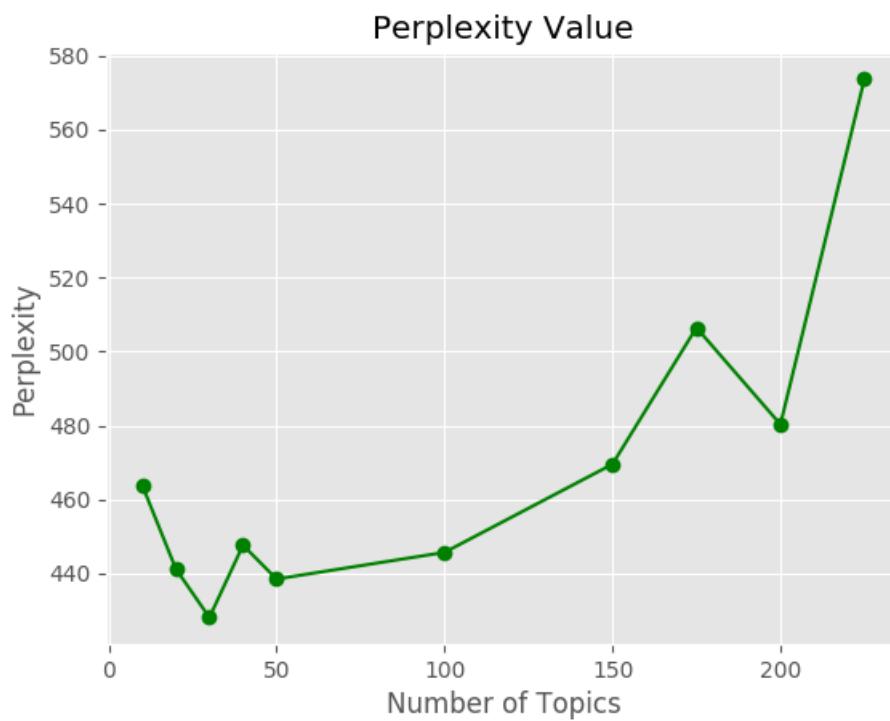
Topic modeling

Learning a latent topic model use LDA is the third step in this scenario. The [gensim](#) Python package is required in this step to learn an [LDA topic model](#). The corresponding notebook for this step is [3_Topic_Model_Training.ipynb](#). You can also refer to [step3.py](#) for how to use the document analysis package.

In this step, the main task is to learn an LDA topic model and tune the hyper parameters. There are multiple parameters need to be tuned when train an LDA model, but the most important parameter is the number of topics. Too few topics may not have insight to the document collection; while choosing too many will result in the "over-clustering" of a corpus into many small, highly similar topics. The purpose of this scenario is to show how to tune the number of topics. Azure Machine Learning Workbench provides the freedom to run experiments with different parameter configuration on different compute contexts.

In the document analysis Python package, a few functions were defined to help the users figure out the best number of topics. The first approach is by evaluating the coherence of the topic model. There are four evaluation matrices supported: [u_mass](#), [c_v](#), [c_uci](#), and [c_npmi](#). The details of those four metrics are discussed in [this paper](#). The second approach is to evaluate the perplexity on a held-out corpus.

For the perplexity evaluation, a 'U' shape curve is expected to find out the best number of topics. And the elbow position is the best choice. It is recommended to evaluate multiple times with different random seed and get the average. The coherence evaluate is expected to be a 'n' shape, which means the coherence increases with increasing the number of topics and then decrease. An example plot of perplexity and [c_v](#) coherence is showing as follows.



In this scenario, the perplexity increases significantly after 200 topics, while the coherence value decreases significantly after 200 topics as well. Based on those graphs and the desire for more general topics versus over clustered topics, choose 200 topics should be a good option.

You can train one LDA topic model in one experiment run, or train and evaluate multiple LDA models with different topic number configurations in a single experiment run. It is recommended to run multiple times for one configuration and then get the average coherence and/or perplexity evaluations. The details of how to use the document analysis package can be found in `step3.py` file. An example code snippet is as follows.

```

topicmodeler = TopicModeler(docs,
    stopWordFile=FUNCTION_WORDS_FILE,
    minWordCount=MIN_WORD_COUNT,
    minDocCount=MIN_DOC_COUNT,
    maxDocFreq=MAX_DOC_FREQ,
    workers=cpu_count()-1,
    numTopics=NUM_TOPICS,
    numIterations=NUM_ITERATIONS,
    passes=NUM_PASSES,
    chunksize=CHUNK_SIZE,
    random_state=RANDOM_STATE,
    test_ratio=test_ratio)

# Train an LDA topic model
lda = topicmodeler.TrainLDA(saveModel=saveModel)

# Evaluate coherence metrics
coherence = topicmodeler.EvaluateCoherence(lda, coherence_types)

# Evaluate perplexity on a held-out corpus
perplex = topicmodeler.EvaluatePerplexity(lda)

```

NOTE

The execution time to train an LDA topic model depends on multiple factors such as the size of corpus, hyper parameter configuration, as well as the number of cores on the machine. Using multiple CPU cores trains a model faster. However, with the same hyper parameter setting more cores means fewer updates during training. It is recommended to have **at least 100 updates to train a converged LDA model**. The relationship between number of updates and hyper parameters is discussed in [this post](#) and [this post](#). In our tests, it took about 3 hours to train an LDA model with 200 topics using the configuration of `workers=15` , `passes=10` , `chunksize=1000` on a machine with 16 cores (2.0 GHz).

Topic summarization and analysis

The topic summarization and analysis consists of two notebooks, while there are no corresponding functions in the document analysis package.

In `4_Topic_Model_Summarization.ipynb`, it shows how to summarize the contents of the document based on a trained LDA topic model. The summarization is applied to an LDA topic model learned in step 3. It shows how to measure the importance or quality of a topic using topic to document purity measure. This purity measure assumes latent topics that dominate the documents in which they appear are more semantically important than latent topics that are weakly spread across many documents. This concept was introduced in the paper "[Latent Topic Modeling for Audio Corpus Summarization](#)."

Notebook `5_Topic_Model_Analysis.ipynb` shows how to analyze the topical content of a collection of documents and correlate topical information against other meta-data associated with the document collection. A few plots are introduced in this notebook to help the users better understand the learned topic and the document collection.

Notebook `6_Interactive_Visualization.ipynb` shows how to interactively visualize learned topic model. It includes four interactive visualization tasks.

Conclusion

This real world scenario highlights how to use well-known text analytics techniques (in this case, phrase learning and LDA topic modeling) to produce a robust model, and how Azure Machine Learning Workbench can help track model performance and seamlessly run learners at higher scale. In more detail:

- Use phrase learning and topic modeling to process a collection of documents and build a robust model. If the collection of documents is huge, Azure Machine Learning Workbench can easily scale it up and out.

Additionally, users have the freedom to run experiments on different compute context easily from within Azure Machine Learning Workbench.

- Azure Machine Learning Workbench provides both options to run notebooks in a step by step manner and Python script to run an entire experiment.
- Hyper-parameter tuning using Azure Machine Learning Workbench to find the best number of topics needed to learn. Azure Machine Learning Workbench can help track the model performance and seamlessly run different learners at higher scale.
- Azure Machine Learning Workbench can manage the run history and learned models. It enables data scientists to quickly identify the best performing models, and to find the scripts and data used to generate them.

References

- **Timothy J. Hazen, Fred Richardson**, *Modeling Multiword Phrases with Constrained Phrases Tree for Improved Topic Modeling of Conversational Speech*. Spoken Language Technology Workshop (SLT), 2012 IEEE. IEEE, 2012.
- **Timothy J. Hazen**, *Latent Topic Modeling for Audio Corpus Summarization*. 12th Annual Conference of the International Speech Communication Association. 2011.
- **Michael Roder, Andreas Both, Alexander Hinneburg**, *Exploring the Space of Topic Coherence Measures*. Proceedings of the eighth ACM international conference on Web search and data mining. ACM, 2015.

Q & A Matching using Azure Machine Learning workbench

9/24/2018 • 7 minutes to read • [Edit Online](#)

NOTE

This article is deprecated. Support for earlier versions of this service will end incrementally. [View the support timeline](#). Start using the latest version with this [quickstart](#).

Answering open ended questions is difficult and often requires manual effort from subject matter experts (SMEs). To help reduce the demands on internal SMEs, companies often create lists of Frequently Asked Questions (FAQs) as a means of assisting users. This example showcases various effective machine learning methods to match open ended queries to pre-existing FAQ question/answers pairs. This example demonstrates an easy development process for building such a solution using the Azure Machine Learning Workbench.

Link to the gallery GitHub repository

<https://github.com/Azure/MachineLearningSamples-QnAMatching>

Overview

This example addresses the problem of mapping user questions to pre-existing Question & Answer (Q&A) pairs as is typically provided in a list of Frequently Asked Questions (that is, a FAQ) or in the Q&A pairs present on websites like [Stack Overflow](#). There are many approaches to match a question to its correct answer, such as finding the answer that is the most similar to the question. However, in this example open ended questions are matched to previously asked questions by assuming that each answer in the FAQ can answer multiple semantically equivalent questions.

The key steps required to deliver this solution are as follows:

1. Clean and process text data.
2. Learn informative phrases, which are multi-word sequences that provide more information when viewed in sequence than when treated independently.
3. Extract features from text data.
4. Train text classification models and evaluate model performance.

Prerequisites

The prerequisites to run this example are as follows:

1. An [Azure account](#) (free trials are available).
2. An installed copy of [Azure Machine Learning Workbench](#) following the [quick start installation guide](#) to install the program and create a workspace.
3. This example could be run on any compute context. However, it is recommended to run it on a multi-core machine with at least of 16-GB memory and 5-GB disk space.

Create a new workbench project

Create a new project using this example as a template:

1. Open Azure Machine Learning Workbench
2. On the **Projects** page, click the + sign and select **New Project**
3. In the **Create New Project** pane, fill in the information for your new project
4. In the **Search Project Templates** search box, type "Q & A Matching" and select the template
5. Click **Create**

Data description

The dataset used in this example is a Stack Exchange Data Dump stored at [archive.org](#). This data is an anonymized dump of all user-contributed content on the [Stack Exchange network](#). Each site in the network is formatted as a separate archive consisting of XML files zipped via 7-zip using bzip2 compression. Each site archive includes Posts, Users, Votes, Comments, PostHistory, and PostLinks.

Data source

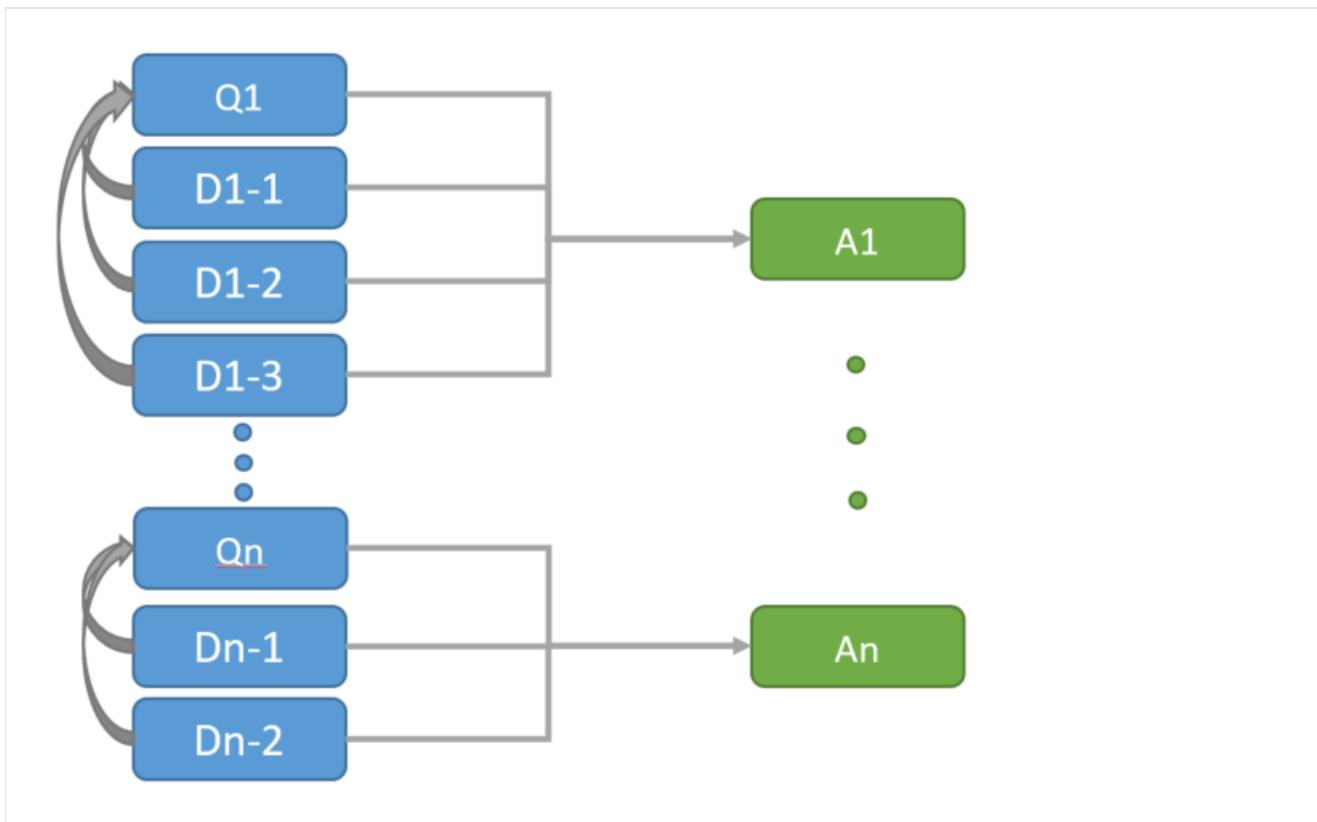
This example uses the [Posts data \(10 GB\)](#) and [PostLinks data \(515 MB\)](#) from the Stack Overflow site. For complete schema information, see the [readme.txt](#).

The `PostTypeId` field in the Posts data indicates whether a post is a `Question` or an `Answer`. The `PostLinkTypeID` field in the PostLinks data indicates whether two posts are linked or duplicate. Question posts typically include some tags, which are keywords that categorize a question with other similar/duplicate questions. There are some tags with high frequency, such as `javascript`, `java`, `c#`, `php` etc., consist of a larger number of question posts. In this example, a subset of Q&A pairs with the `javascript` tag is extracted.

Additionally, a question post may be related to multiple answer posts or duplicate question posts. To construct a list of FAQ from these two datasets, some data collection criteria are considered. The three sets of compiled data are selected using a SQL script, which is not included in this example. The resulting data description is as follows:

- `Original Questions (Q)` : Question posts are asked and answered on Stack Overflow site.
- `Duplications (D)` : Question posts duplicate other pre-existing questions (`PostLinkTypeID = 3`), which are the original questions. Duplications are considered as semantically equivalent to the original questions in the sense that the answer provided to the original question also answers the new duplicate question.
- `Answers (A)` : Each original question and its duplications may link to more than one answer posts. This example only selects the answer post that is either accepted by the original author (filtered by `AcceptedAnswerId`) or has the highest score (ranked by `Score`) in the original questions.

The combination of these three datasets creates Q&A pairs where an answer (A) is mapped to one original question (Q) and multiple duplicate questions (D) as illustrated by the following data diagram:



Data structure

The data schema and direct download links of the three datasets can be found in the following table:

DATASET	FIELD	TYPE	DESCRIPTION
questions	Id	String	The unique question ID (primary key)
	AnswerId	String	The unique answer ID per question
	Text0	String	The raw text data including the question's title and body
	CreationDate	Timestamp	The timestamp of when the question has been asked
dupes	Id	String	The unique duplication ID (primary key)
	AnswerId	String	The answer ID associated with the duplication
	Text0	String	The raw text data including the duplication's title and body
	CreationDate	Timestamp	The timestamp of when the duplication has been asked
answers	Id	String	The unique answer ID (primary key)

Dataset	Field	Type	Description
	text0	String	The raw text data of the answer

Scenario structure

The Q&A matching example is presented through three types of files. The first type is a series of Jupyter Notebooks that show the step-by-step descriptions of the entire workflow. The second type is a set of Python files contain custom Python modules for phrase learning and feature extraction. These Python modules are generic enough to not only serve this example but also other use cases. The third type is a set of Python files to tune hyper-parameters and track model performance using the Azure Machine Learning Workbench.

The files in this example are organized as follows.

File Name	Type	Description
Image	Folder	The folder used to save images for the README file
notebooks	Folder	The Jupyter Notebooks folder
modules	Folder	The Python modules folder
scripts	Folder	The Python files folder
notebooks/Part_1_Data_Preparation.ipynb	Jupyter Notebook	Access the sample data, pre-process the text, and prepare training and test datasets
notebooks/Part_2_Phrase_Learning.ipynb	Jupyter Notebook	Learn informative phrases and tokenize text into Bag-of-Words (BOWs) representations
notebooks/Part_3_Model_Training_and_Evaluation.ipynb	Jupyter Notebook	Extract features, train text classification models, and evaluation model performance
modules/__init__.py	Python file	The Python package init file
modules/phrase_learning.py	Python file	The Python modules used to transform the raw data and learn informative phrases
modules/feature_extractor.py	Python file	The Python modules extract features for model training
scripts/naive_bayes.py	Python file	The Python to tune hyper-parameters in the Naive Bayes model
scripts/random_forest.py	Python file	The Python to tune hyper-parameters in the Random Forest model
README.md	Markdown file	The README markdown file

NOTE

The series of notebooks is built under Python 3.5.

Data ingestion and transformation

The three compiled datasets are stored in a Blob storage and are retrieved in [Part_1_Data_Preparation.ipynb](#) notebook.

Before training the text classification models, the text in the questions is cleaned and preprocessed to exclude code snippets. An unsupervised phrase learning is applied over the training material to learn informative multi-word sequences. These phrases are represented as single compound word units in the downstream Bag-of-Words (BOWs) featurization used by the text classification models.

The detailed step-by-step descriptions of text preprocessing and phrase learning can be found in the Notebooks [Part_1_Data_Preparation.ipynb](#) and [Part_2_Phase_Learning.ipynb](#), respectively.

Modeling

This example is designed to score new questions against the pre-existing Q&A pairs by training text classification models where each pre-existing Q&A pair is a unique class and a subset of the duplicate questions for each Q&A pair are available as training material. In the example, the original questions and 75% of the duplicate questions are retained for training and the most recently posted 25% of duplicate questions are held-out as evaluation data.

The classification model uses an ensemble method to aggregate three base classifiers, including **Naive Bayes**, **Support Vector Machine**, and **Random Forest**. In each base classifier, the `AnswerId` is used as the class label and the BOWs representations is used as the features.

The model training process is illustrated in [Part_3_Model_Training_and_Evaluation.ipynb](#).

Evaluation

Two different evaluation metrics are used to assess performance.

1. `Average Rank (AR)` : indicates the average position where the correct answer is found in the list of retrieved Q&A pairs (out of the full set of 103 answer classes).
2. `Top 3 Percentage` : indicates the percentage of test questions that the correct answer can be retrieved in the top three choices in the returned ranked list.

The evaluation is demonstrated in [Part_3_Model_Training_and_Evaluation.ipynb](#).

Conclusion

This example highlights how to use well-known text analytics techniques, such as phrase learning and text classification, to produce a robust model. It also showcases how Azure Machine Learning Workbench can help with interactive solution development and track model performance.

Some key highlights of this example are:

- The question and answer matching problem can be effectively solved with phrase learning and text classification models.
- Demonstrating interactive model development with Azure Machine Learning Workbench and Jupyter Notebook.
- Azure Machine Learning Workbench manages the run history and learned models with logging the evaluation metrics for comparison purposes. These features enables quick hyper-parameter tuning and helps identify the best performing models.

References

Timothy J. Hazen, Fred Richardson, *Modeling Multiword Phrases with Constrained Phrases Tree for Improved Topic Modeling of Conversational Speech*. Spoken Language Technology Workshop (SLT), 2012 IEEE. IEEE, 2012.

Timothy J. Hazen, *MCE Training Techniques for Topic Identification of Spoken Audio Documents* in IEEE Transactions on Audio, Speech, and Language Processing, vol. 19, no. 8, pp. 2451-2460, Nov. 2011.

Predictive maintenance for real-world scenarios

12/4/2018 • 7 minutes to read • [Edit Online](#)

NOTE

This article is deprecated. Support for earlier versions of this service will end incrementally. [View the support timeline](#). Start using the latest version with this [quickstart](#).

The impact of unscheduled equipment downtime can be detrimental for any business. It's critical to keep field equipment running to maximize utilization and performance, and to minimize costly, unscheduled downtime. Early identification of issues can help allocate limited maintenance resources in a cost-effective way and enhance quality and supply chain processes.

This scenario explores a relatively [large-scale simulated data set](#) to walk through a predictive maintenance data science project from data ingestion, feature engineering, model building, and model operationalization and deployment. The code for the entire process is written in the Jupyter Notebook by using PySpark in Azure Machine Learning Workbench. The final model is deployed by using Azure Machine Learning Model Management to make real-time equipment failure predictions.

Cortana Intelligence Gallery GitHub repository

The Cortana Intelligence Gallery for the PM tutorial is a public GitHub repository (<https://github.com/Azure/MachineLearningSamples-PredictiveMaintenance>) where you can report issues and make contributions.

Use case overview

A major problem that's faced by businesses in asset-heavy industries is the significant costs that are associated with delays due to mechanical problems. Most businesses are interested in predicting when these problems might arise to proactively prevent them before they occur. The goal is to reduce the costs by reducing downtime and possibly increase safety.

This scenario takes ideas from the [Predictive maintenance playbook](#) to demonstrate how to build a predictive model for a simulated data set. The example data is derived from common elements that are observed in many predictive maintenance use cases.

The business problem for this simulated data is to predict issues that are caused by component failures. The business question is "*What's the probability that a machine goes down due to failure of a component?*" This problem is formatted as a multi-class classification problem (multiple components per machine). A machine learning algorithm is used to create the predictive model. The model is trained on historical data that's collected from machines. In this scenario, the user goes through various steps to implement the model in the Machine Learning Workbench environment.

Prerequisites

- An [Azure account](#) (free trials are available).
- An installed copy of [Azure Machine Learning Workbench](#). Follow the [Quickstart installation guide](#) to install the program and create a workspace.
- Azure Machine Learning Operationalization requires a local deployment environment and an [Azure Machine Learning Model Management account](#).

This example runs on any Machine Learning Workbench compute context. However, it's recommended to run the example with at least 16 GB of memory. This scenario was built and tested on a Windows 10 machine running a remote DS4_V2 standard [Data Science Virtual Machine \(DSVM\) for Linux \(Ubuntu\)](#).

Model operationalization was done by using version 0.1.0a22 of the Azure Machine Learning CLI.

Create a new Workbench project

Create a new project by using this example as a template:

1. Open Machine Learning Workbench.
2. On the **Projects** page, select **+**, and then select **New Project**.
3. In the **Create New Project** pane, fill in the information for your new project.
4. In the **Search Project Templates** search box, type "Predictive Maintenance" and select the **Predictive Maintenance** template.
5. Select **Create**.

Prepare the notebook server computation target

To run on your local machine, from the Machine Learning Workbench **File** menu, select either **Open Command Prompt** or **Open PowerShell CLI**. The CLI interface allows you to access your Azure services by using the `az` commands. First, log in to your Azure account with the command:

```
az login
```

This command provides an authentication key to use with the <https://aka.ms/devicelogin> URL. The CLI waits until the device login operation returns and provides some connection information. Next, if you have a local [Docker](#) installation, prepare the local compute environment with the command:

```
az ml experiment prepare --target docker --run-configuration docker
```

It's preferable to run on a [DSVM for Linux \(Ubuntu\)](#) for memory and disk requirements. After the DSVM is configured, prepare the remote Docker environment with the following two commands:

```
az ml computetarget attach remotedocker --name [Connection_Name] --address [VM_IP_Address] --username [VM_Username] --password [VM_UserPassword]
```

After you're connected to the remote Docker container, prepare the DSVM Docker compute environment with the command:

```
az ml experiment prepare --target [Connection_Name] --run-configuration [Connection_Name]
```

With the Docker compute environment prepared, open the Jupyter notebook server from the Azure Machine Learning Workbench **Notebooks** tab, or start a browser-based server with the command:

```
az ml notebook start
```

The example notebooks are stored in the Code directory. The notebooks are set up to run sequentially, starting on the first (Code\1_data_ingestion.ipynb) notebook. When you open each notebook, you're prompted to select the compute kernel. Choose the [Project_Name]_Template [Connection_Name] kernel to execute on the previously configured DSVM.

Data description

The [simulated data](#) consists of the following [five comma-separated values \(.csv\) files](#):

- **Machines:** Features that differentiate each machine, such as age and model.
- **Errors:** The error log contains non-breaking errors that are thrown while the machine is still operational. These errors are not considered failures, though they can be predictive of a future failure event. The date-time values for the errors are rounded to the closest hour since the telemetry data is collected at an hourly rate.
- **Maintenance:** The maintenance log contains both scheduled and unscheduled maintenance records. Scheduled maintenance corresponds with the regular inspection of components. Unscheduled maintenance can arise from mechanical failure or other performance degradation. The date-time values for maintenance are rounded to the closest hour since the telemetry data is collected at an hourly rate.
- **Telemetry:** The telemetry data consists of time series measurements from multiple sensors within each machine. The data is logged by averaging sensor values over each one hour interval.
- **Failures:** Failures correspond to component replacements within the maintenance log. Each record contains the Machine ID, component type, and replacement date and time. These records are used to create the machine learning labels that the model is trying to predict.

To download the raw data sets from the GitHub repository, and create the PySpark data sets for this analysis, see the [Data Ingestion Jupyter Notebook scenario](#) in the [Code](#) folder.

Scenario structure

The content for the scenario is available at the [GitHub repository](#).

The [Readme](#) file outlines the workflow from preparing the data, building a model, and then deploying a solution for production. Each step of the workflow is encapsulated in a Jupyter notebook in the [Code](#) folder within the repository.

[Code\1_data_ingestion.ipynb](#): This notebook downloads the five input .csv files, and does some preliminary data cleanup and visualization. The notebook converts each data set to PySpark format and stores it in an Azure blob container for use in the Feature Engineering Notebook.

[Code\2_feature_engineering.ipynb](#): The model features are constructed from the raw data set from Azure Blob storage by using a standard time series approach for telemetry, errors, and maintenance data. The failure-related component replacements are used to construct the model labels that describe which component failed. The labeled feature data is saved in an Azure blob for the Model Building Notebook.

[Code\3_model_building.ipynb](#): The Model Building Notebook uses the labeled feature data set and splits the data into train and dev data sets along the date-time stamp. The notebook is set up as an experiment with pyspark.ml.classification models. The training data is vectorized. The user can experiment with either a **DecisionTreeClassifier** or **RandomForestClassifier** to manipulate hyperparameters to find the best performing model. Performance is determined by evaluating measurement statistics on the dev data set. These statistics are logged back in to the Machine Learning Workbench runtime screen for tracking. At each run, the notebook saves the resulting model to the local disk that's running the Jupyter notebook kernel.

[Code\4_operationalization.ipynb](#): This notebook uses the last model that's saved to the local (Jupyter notebook kernel) disk to build the components for deploying the model into an Azure web service. The full operational assets are compacted into the o16n.zip file that's stored in another Azure blob container. The zipped file contains:

- **service_schema.json**: The schema definition file for the deployment.
- **pdmsscore.py**: The **init()** and **run()** functions that are required by the Azure web service.
- **pdmrfull.model**: The model definition directory.

The notebook tests the functions with the model definition before packaging the operationalization assets for

deployment. Instructions for deployment are included at the end of the notebook.

Conclusion

This scenario gives an overview of building an end-to-end predictive maintenance solution by using PySpark within the Jupyter Notebook environment in Machine Learning Workbench. This example scenario also details model deployment through the Machine Learning Model Management environment to make real-time equipment failure predictions.

References

The following references provide examples of other predictive maintenance use cases for various platforms:

- [Predictive Maintenance Solution Template](#)
- [Predictive Maintenance Modeling Guide](#)
- [Predictive Maintenance Modeling Guide using SQL R Services](#)
- [Predictive Maintenance Modeling Guide Python Notebook](#)
- [Predictive Maintenance using PySpark](#)
- [Deep learning for predictive maintenance](#)

Next steps

Other example scenarios are available in Machine Learning Workbench that demonstrate additional features of the product.

Aerial Image Classification

11/7/2018 • 21 minutes to read • [Edit Online](#)

NOTE

This article is deprecated. Support for earlier versions of this service will end incrementally. [View the support timeline](#). Start using the latest version with this [quickstart](#).

This example demonstrates how to use Azure Machine Learning Workbench to coordinate distributed training and operationalization of image classification models. We use two approaches for training: (i) refining a deep neural network using an [Azure Batch AI](#) GPU cluster, and (ii) using the [Microsoft Machine Learning for Apache Spark \(MMLSpark\)](#) package to featurize images using pretrained CNTK models and to train classifiers using the derived features. We then apply the trained models in parallel fashion to large image sets in the cloud using an [Azure HDInsight Spark](#) cluster, allowing us to scale the speed of training and operationalization by adding or removing worker nodes.

This example highlights two approaches to transfer learning, which leverages pretrained models to avoid the costs of training deep neural networks from scratch. Retraining a deep neural network typically requires GPU compute, but can lead to greater accuracy when the training set is sufficiently large. Training a simple classifier on featurized images does not require GPU compute, is inherently fast and arbitrarily scalable, and fits fewer parameters. This method is therefore an excellent choice when few training samples are available -- as is often the case for custom use cases.

The Spark cluster used in this example has 40 worker nodes and costs ~\$40/hr to operate; the Batch AI cluster resources include eight GPUs and cost ~\$10/hr to operate. Completing this walkthrough takes approximately three hours. When you are finished, follow the cleanup instructions to remove the resources you have created and stop incurring charges.

Link to the Gallery GitHub repository

The public GitHub repository for this real world scenario contains all materials, including code samples, needed for this example:

<https://github.com/Azure/MachineLearningSamples-AerialImageClassification>

Use case description

In this scenario, we train machine learning models to classify the type of land shown in aerial images of 224-meter x 224-meter plots. Land use classification models can be used to track urbanization, deforestation, loss of wetlands, and other major environmental trends using periodically collected aerial imagery. We have prepared training and validation image sets based on imagery from the U.S. National Agriculture Imagery Program and land use labels published by the U.S. National Land Cover Database. Example images in each land use class are shown here:



Developed



Barren



Forested



Grassland



Shrub



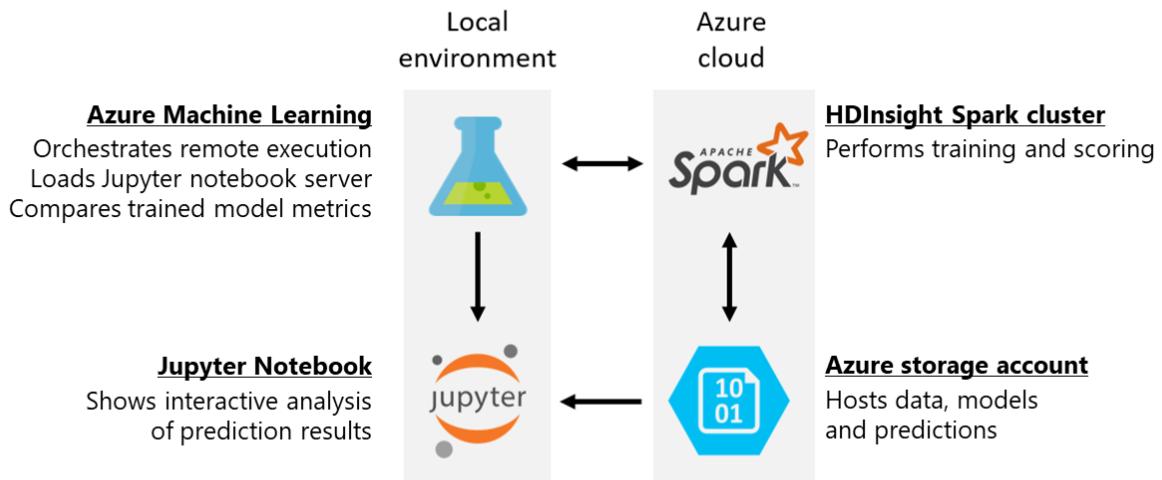
Cultivated

After training and validating the classifier model, we will apply it to aerial images spanning Middlesex County, MA -- home of Microsoft's New England Research & Development (NERD) Center -- to demonstrate how these models can be used to study trends in urban development.

To produce an image classifier using transfer learning, data scientists often construct multiple models with a range of methods and select the most performant model. Azure Machine Learning Workbench can help data scientists coordinate training across different compute environments, track and compare the performance of multiple models, and apply a chosen model to large datasets on the cloud.

Scenario structure

In this example, image data and pretrained models are housed in an Azure storage account. An Azure HDInsight Spark cluster reads these files and constructs an image classification model using MMLSpark. The trained model and its predictions are then written to the storage account, where they can be analyzed and visualized by a Jupyter notebook running locally. Azure Machine Learning Workbench coordinates remote execution of scripts on the Spark cluster. It also tracks accuracy metrics for multiple models trained using different methods, allowing the user to select the most performant model.



These step-by-step instructions begin by guiding you through the creation and preparation of an Azure storage account and Spark cluster, including data transfer and dependency installation. They then describe how to launch training jobs and compare the performance of the resulting models. Finally, they illustrate how to apply a chosen model to a large image set on the Spark cluster and analyze the prediction results locally.

Set up the execution environment

The following instructions guide you through the process of setting up execution environment for this example.

Prerequisites

- An [Azure account](#) (free trials are available)
 - You will create an HDInsight Spark cluster with 40 worker nodes (168 cores total). Ensure that your account has enough available cores by reviewing the "Usage + quotas" tab for your subscription in Azure portal.
 - If you have fewer cores available, you may modify the HDInsight cluster template to decrease the number of workers provisioned. Instructions for this appear under the "Create the HDInsight Spark cluster" section.
 - This sample creates a Batch AI Training cluster with two NC6 (1 GPU, 6 vCPU) VMs. Ensure that your account has enough available cores in the East US region by reviewing the "Usage + quotas" tab for your subscription in Azure portal.
- [Azure Machine Learning Workbench](#)
 - Follow the [Install and create Quickstart](#) to install the Azure Machine Learning Workbench and create Experimentation and Model Management Accounts.
- [Batch AI Python SDK and Azure CLI](#)
 - Complete the following sections in the [Batch AI Recipes README](#):
 - "Prerequisites"
 - "Create and get your Azure Active Directory (AAD) application"
 - "Register BatchAI Resource Providers" (under "Run Recipes Using Azure CLI")
 - "Install Azure Batch AI Management Client"
 - "Install Azure Python SDK"
 - Record the client ID, secret, and tenant ID of the Azure Active Directory application you are directed to create. You will use those credentials later in this tutorial.

- As of this writing, Azure Machine Learning Workbench and Azure Batch AI use separate forks of the Azure CLI. For clarity, we refer to the Workbench's version of the CLI as "a CLI launched from Azure Machine Learning Workbench" and the general-release version (which includes Batch AI) as "Azure CLI."
- [AzCopy](#), a free utility for coordinating file transfer between Azure storage accounts
 - Ensure that the folder containing the AzCopy executable is on your system's PATH environment variable. (Instructions on modifying environment variables are available [here](#).)
- An SSH client; we recommend [PuTTY](#).

This example was tested on a Windows 10 PC; you should be able to run it from any Windows machine, including Azure Data Science Virtual Machines. The Azure CLI was installed from an MSI according to [these instructions](#). Minor modifications may be required (for example, changes to filepaths) when running this example on macOS.

Set up Azure resources

This example requires an HDInsight Spark cluster and an Azure storage account to host relevant files. Follow these instructions to create these resources in a new Azure resource group:

Create a new Workbench project

Create a new project using this example as a template:

1. Open Azure Machine Learning Workbench
2. On the **Projects** page, click the + sign and select **New Project**
3. In the **Create New Project** pane, fill in the information for your new project
4. In the **Search Project Templates** search box, type "Aerial Image Classification" and select the template
5. Click **Create**

Create the resource group

1. After loading your project in Azure Machine Learning Workbench, open a Command Line Interface (CLI) by clicking File -> Open Command Prompt. Use this version of the CLI for the majority of the tutorial. (Where indicated, you are asked to open another version of the CLI to prepare Batch AI resources.)
2. From the command-line interface, log in to your Azure account by running the following command:

```
az login
```

You are asked to visit a URL and type in a provided temporary code; the website requests your Azure account credentials.

3. When login is complete, return to the CLI and run the following command to determine which Azure subscriptions are available to your Azure account:

```
az account list
```

This command lists all subscriptions associated with your Azure account. Find the ID of the subscription you would like to use. Write the subscription ID where indicated in the following command, then set the active subscription by executing the command:

```
az account set --subscription [subscription ID]
```

4. The Azure resources created in this example are stored together in an Azure resource group. Choose a unique resource group name and write it where indicated, then execute both commands to create the Azure resource group:

```
set AZURE_RESOURCE_GROUP=[resource group name]
az group create --location eastus --name %AZURE_RESOURCE_GROUP%
```

Create the storage account

We now create the storage account that hosts project files that must be accessed by the HDInsight Spark.

1. Choose a unique storage account name and write it where indicated in the following `set` command, then create an Azure storage account by executing both commands:

```
set STORAGE_ACCOUNT_NAME=[storage account name]
az storage account create --name %STORAGE_ACCOUNT_NAME% --resource-group %AZURE_RESOURCE_GROUP% --sku Standard_LRS
```

2. List the storage account keys by running the following command:

```
az storage account keys list --resource-group %AZURE_RESOURCE_GROUP% --account-name %STORAGE_ACCOUNT_NAME%
```

Record the value of `key1` as the storage key in the following command, then run the command to store the value.

```
set STORAGE_ACCOUNT_KEY=[storage account key]
```

3. Create a file share named `baitshare` in your storage account with the following command:

```
az storage share create --account-name %STORAGE_ACCOUNT_NAME% --account-key %STORAGE_ACCOUNT_KEY% --name baitshare
```

4. In your favorite text editor, load the `settings.cfg` file from the Azure Machine Learning Workbench project's "Code" subdirectory, and insert the storage account name and key as indicated. Save and close the `settings.cfg` file.
5. If you have not already done so, download and install the [AzCopy](#) utility. Ensure that the AzCopy executable is on your system path by typing "AzCopy" and pressing Enter to show its documentation.
6. Issue the following commands to copy all of the sample data, pretrained models, and model training scripts to the appropriate locations in your storage account:

```

AzCopy /Source:https://mawahsparktutorial.blob.core.windows.net/test /SourceSAS:"?sv=2017-04-
17&ss=bf&srt=sco&sp=rwl&se=2037-08-25T22:02:55Z&st=2017-08-
25T14:02:55Z&spr=https,http&sig=yy06fyau9ilAeW7TpkgbAqeTnrPR%2BpP1eh9TcpIXWw%3D"
/Dest:https://%STORAGE_ACCOUNT_NAME%.blob.core.windows.net/test /DestKey:%STORAGE_ACCOUNT_KEY% /S
AzCopy /Source:https://mawahsparktutorial.blob.core.windows.net/train /SourceSAS:"?sv=2017-04-
17&ss=bf&srt=sco&sp=rwl&se=2037-08-25T22:02:55Z&st=2017-08-
25T14:02:55Z&spr=https,http&sig=yy06fyau9ilAeW7TpkgbAqeTnrPR%2BpP1eh9TcpIXWw%3D"
/Dest:https://%STORAGE_ACCOUNT_NAME%.blob.core.windows.net/train /DestKey:%STORAGE_ACCOUNT_KEY% /S
AzCopy /Source:https://mawahsparktutorial.blob.core.windows.net/middlesexma2016 /SourceSAS:"?sv=2017-04-
17&ss=bf&srt=sco&sp=rwl&se=2037-08-25T22:02:55Z&st=2017-08-
25T14:02:55Z&spr=https,http&sig=yy06fyau9ilAeW7TpkgbAqeTnrPR%2BpP1eh9TcpIXWw%3D"
/Dest:https://%STORAGE_ACCOUNT_NAME%.blob.core.windows.net/middlesexma2016
/DestKey:%STORAGE_ACCOUNT_KEY% /S
AzCopy /Source:https://mawahsparktutorial.blob.core.windows.net/pretrainedmodels /SourceSAS:"?sv=2017-
04-17&ss=bf&srt=sco&sp=rwl&se=2037-08-25T22:02:55Z&st=2017-08-
25T14:02:55Z&spr=https,http&sig=yy06fyau9ilAeW7TpkgbAqeTnrPR%2BpP1eh9TcpIXWw%3D"
/Dest:https://%STORAGE_ACCOUNT_NAME%.blob.core.windows.net/pretrainedmodels
/DestKey:%STORAGE_ACCOUNT_KEY% /S
AzCopy /Source:https://mawahsparktutorial.blob.core.windows.net/pretrainedmodels /SourceSAS:"?sv=2017-
04-17&ss=bf&srt=sco&sp=rwl&se=2037-08-25T22:02:55Z&st=2017-08-
25T14:02:55Z&spr=https,http&sig=yy06fyau9ilAeW7TpkgbAqeTnrPR%2BpP1eh9TcpIXWw%3D"
/Dest:https://%STORAGE_ACCOUNT_NAME%.file.core.windows.net/baitshare/pretrainedmodels
/DestKey:%STORAGE_ACCOUNT_KEY% /S
AzCopy /Source:https://mawahsparktutorial.blob.core.windows.net/scripts /SourceSAS:"?sv=2017-04-
17&ss=bf&srt=sco&sp=rwl&se=2037-08-25T22:02:55Z&st=2017-08-
25T14:02:55Z&spr=https,http&sig=yy06fyau9ilAeW7TpkgbAqeTnrPR%2BpP1eh9TcpIXWw%3D"
/Dest:https://%STORAGE_ACCOUNT_NAME%.file.core.windows.net/baitshare/scripts
/DestKey:%STORAGE_ACCOUNT_KEY% /S

```

Expect file transfer to take around one hour. While you wait, you can proceed to the following section: you may need to open another Command Line Interface through Workbench and redefine the temporary variables there.

Create the HDInsight Spark cluster

Our recommended method to create an HDInsight cluster uses the HDInsight Spark cluster resource manager template included in the "Code\01_Data_Acquisition_and_Understanding\01_HDInsight_Spark_Provisioning" subfolder of this project.

- The HDInsight Spark cluster template is the "template.json" file under the "Code\01_Data_Acquisition_and_Understanding\01_HDInsight_Spark_Provisioning" subfolder of this project. By default, the template creates a Spark cluster with 40 worker nodes. If you must adjust that number, open the template in your favorite text editor and replace any instances of "40" with the worker node number of your choice.
 - You may encounter out-of-memory errors later if the number of worker nodes you choose is smaller. To combat memory errors, you may run the training and operationalization scripts on a subset of the available data as described later in this document.
- Choose a unique name and password for the HDInsight cluster and write them where indicated in the following command: Then create the cluster by issuing the commands:

```

set HDINSIGHT_CLUSTER_NAME=[HDInsight cluster name]
set HDINSIGHT_CLUSTER_PASSWORD=[HDInsight cluster password]
az group deployment create --resource-group %AZURE_RESOURCE_GROUP% --name hdispark --template-file
"Code\01_Data_Acquisition_and_Understanding\01_HDInsight_Spark_Provisioning\template.json" --parameters
storageAccountName=%STORAGE_ACCOUNT_NAME%.blob.core.windows.net storageAccountKey=%STORAGE_ACCOUNT_KEY%
clusterName=%HDINSIGHT_CLUSTER_NAME% clusterLoginPassword=%HDINSIGHT_CLUSTER_PASSWORD%

```

Your cluster's deployment may take up to 30 minutes (including provisioning and script action execution).

Set up Batch AI resources

While you wait for your Storage account file transfer and Spark cluster deployment to complete, you can prepare the Batch AI Network File Server (NFS) and GPU cluster. Open an Azure CLI command prompt and run the following command:

```
az --version
```

Confirm that `batchai` is listed among the installed modules. If not, you may be using a different Command Line Interface (for example, one opened through Workbench).

Next, check that provider registration has successfully completed. (Provider registration takes up to fifteen minutes and may still be ongoing if you recently completed the [Batch AI setup instructions](#).) Confirm that both "Microsoft.Batch" and "Microsoft.BatchAI" appear with status "Registered" in the output of the following command:

```
az provider list --query "[].{Provider:namespace, Status:registrationState}" --out table
```

If not, run the following provider registration commands and wait ~15 minutes for registration to complete.

```
az provider register --namespace Microsoft.Batch  
az provider register --namespace Microsoft.BatchAI
```

Modify the following commands to replace the bracketed expressions with the values you used earlier during resource group and storage account creation. Then, store the values as variables by executing the commands:

```
az account set --subscription [subscription ID]  
set AZURE_RESOURCE_GROUP=[resource group name]  
set STORAGE_ACCOUNT_NAME=[storage account name]  
set STORAGE_ACCOUNT_KEY=[storage account key]  
az configure --defaults location=eastus  
az configure --defaults group=%AZURE_RESOURCE_GROUP%
```

Identify the folder containing your Azure Machine Learning project (for example, `C:\Users\<your username>\AzureML\airlineimageclassification`). Replace the bracketed value with the folder's filepath (with no trailing backslash) and execute the command:

```
set PATH_TO_PROJECT=[The filepath of your project's root directory]
```

You are now ready to create the Batch AI resources needed for this tutorial.

Prepare the Batch AI Network File Server

Your Batch AI cluster accesses your training data on a Network File Server. Data access may be several-fold faster when accessing files from an NFS vs. an Azure File Share or Azure Blob Storage.

1. Issue the following command to create a Network File Server:

```
az batchai file-server create -n landuseclassifier -u demoUser -p "Dem0Pa$$w0rd" --vm-size Standard_DS2_V2 --disk-count 1 --disk-size 1000 --storage-sku Premium_LRS
```

2. Check the provisioning status of your Network File Server using the following command:

```
az batchai file-server list
```

When the "provisioningState" of the Network File Server named "landuseclassifier" is "succeeded", it is

ready for use. Expect provisioning to take about five minutes.

3. Find the IP address of your NFS in the output of the previous command (the "fileServerPublicIp" property under "mountSettings"). Write the IP address where indicated in the following command, then store the value by executing the command:

```
set AZURE_BATCH_AI_TRAINING_NFS_IP=[your NFS IP address]
```

4. Using your favorite SSH tool (the following sample command uses [PuTTY](#)), execute this project's `prep_nfs.sh` script on the NFS to transfer the training and validation image sets there.

```
putty -ssh demoUser@%AZURE_BATCH_AI_TRAINING_NFS_IP% -pw Dem0Pa$$w0rd -m %PATH_TO_PROJECT%\Code\01_Data_Acquisition_and_Understanding\02_Batch_AI_Training_Provisioning\prep_nfs.sh
```

Do not be concerned if the data download and extraction progress updates scroll across the shell window so quickly that they are illegible.

If desired, you can confirm that the data transfer has proceeded as planned by logging in to the file server with your favorite SSH tool and checking the `/mnt/data` directory contents. You should find two folders, `training_images` and `validation_images`, each containing with subfolders named according to land use categories. The training and validation sets should contain ~44k and ~11k images, respectively.

Create a Batch AI cluster

1. Create the cluster by issuing the following command:

```
az batchai cluster create -n landuseclassifier2 -u demoUser -p "Dem0Pa$$w0rd" --afs-name baitshare --nfs landuseclassifier --image UbuntuDSVM --vm-size STANDARD_NC6 --max 2 --min 2 --storage-account-name %STORAGE_ACCOUNT_NAME%
```

2. Use the following command to check your cluster's provisioning status:

```
az batchai cluster list
```

When the allocation state for the cluster named "landuseclassifier" changes from resizing to steady, it's possible to submit jobs. However, jobs do not start running until all VMs in the cluster have left the "preparing" state. If the "errors" property of the cluster is not null, an error occurred during cluster creation and it should not be used.

Record Batch AI Training credentials

While you wait for cluster allocation to complete, open the `settings.cfg` file from the "Code" subdirectory of this project in the text editor of your choice. Update the following variables with your own credentials:

- `bait_subscription_id` (your 36-character Azure subscription ID)
- `bait_aad_client_id` (the Azure Active Directory application/client ID mentioned in the "Prerequisites" section)
- `bait_aad_secret` (the Azure Active Directory application secret mentioned in the "Prerequisites" section)
- `bait_aad_tenant` (the Azure Active Directory tenant ID mentioned in the "Prerequisites" section)
- `bait_region` (as of this writing, the only option is: eastus)
- `bait_resource_group_name` (the resource group you chose earlier)

Once you have assigned these values, the modified lines of your `settings.cfg` file should resemble the following text:

```
[Settings]
# Credentials for the Azure Storage account
storage_account_name = yoursaname
storage_account_key = kpIXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXQ==

# Batch AI training credentials
bait_subscription_id = 0caXXXXX-XXXX-XXXX-XXXX-XXXXXXXX9c3
bait_aad_client_id = d0aXXXXX-XXXX-XXXX-XXXX-XXXXXXXX7f8
bait_aad_secret = ygSXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX6I=
bait_aad_tenant = 72fXXXXX-XXXX-XXXX-XXXX-XXXXXXXXXb47
bait_region = eastus
bait_resource_group_name = yourrgname
```

Save and close `settings.cfg`.

You may now close the CLI window where you executed the Batch AI resource creation commands. All further steps in this tutorial use a CLI launched from Azure Machine Learning Workbench.

Prepare the Azure Machine Learning Workbench execution environment

Register the HDInsight cluster as an Azure Machine Learning Workbench compute target

Once HDInsight cluster creation is complete, register the cluster as a compute target for your project as follows:

1. Issue the following command from the Azure Machine Learning Command Line Interface:

```
az ml computetarget attach cluster --name myhdi --address %HDINSIGHT_CLUSTER_NAME%
ssh.azurehdinsight.net --username sshuser --password %HDINSIGHT_CLUSTER_PASSWORD%
```

This command adds two files, `myhdi.runconfig` and `myhdi.compute`, to your project's `aml_config` folder.

2. Open the `myhdi.compute` file in your favorite text editor. Modify the `yarnDeployMode: cluster` line to read `yarnDeployMode: client`, then save and close the file.
3. Run the following command to prepare your HDInsight environment for use:

```
az ml experiment prepare -c myhdi
```

Install local dependencies

Open a CLI from Azure Machine Learning Workbench and install dependencies needed for local execution by issuing the following command:

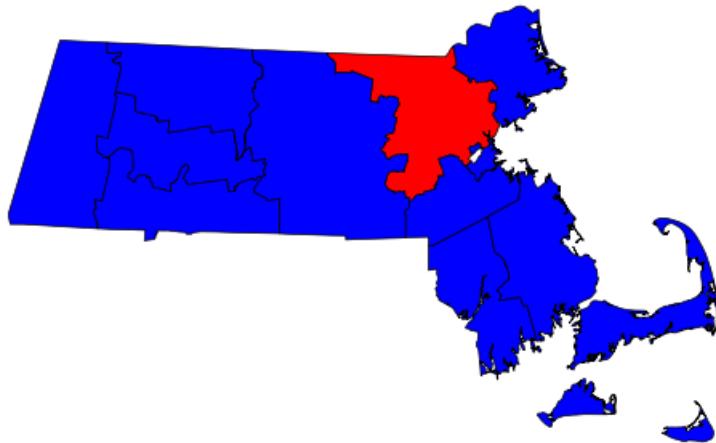
```
pip install matplotlib azure-storage==0.36.0 pillow scikit-learn azure-mgmt-batchai
```

Data acquisition and understanding

This scenario uses publicly available aerial imagery data from the [National Agriculture Imagery Program](#) at 1-meter resolution. We have generated sets of 224 pixel x 224 pixel PNG files cropped from the original NAIP data and sorted according to land use labels from the [National Land Cover Database](#). A sample image with label "Developed" is shown at full size:



Class-balanced sets of ~44k and 11k images are used for model training and validation, respectively. We demonstrate model deployment on a ~67k image set tiling Middlesex County, MA -- home of Microsoft's New England Research and Development (NERD) center. For more information on how these image sets were constructed, see the [Embarrassingly Parallel Image Classification git repository](#).



During setup, the aerial image sets used in this example were transferred to the storage account that you created. The training, validation, and operationalization images are all 224 pixel x 224 pixel PNG files at a resolution of one pixel per square meter. The training and validation images have been organized into subfolders based on their land use label. (The land use labels of the operationalization images are unknown and in many cases ambiguous; some of these images contain multiple land types.) For more information on how these image sets were constructed, see the [Embarrassingly Parallel Image Classification git repository](#).

To view example images in your Azure storage account (optional):

1. Log in to the [Azure portal](#).
2. Search for the name of your storage account in the search bar along the top of your screen. Click on your storage account in the search results.
3. Click on the "Blobs" link in the storage account's main pane.
4. Click on the container named "train." You should see a list of directories named according to land use.
5. Click on any of these directories to load the list of images it contains.
6. Click on any image and download it to view the image.
7. If desired, click on the containers named "test" and "middlesexma2016" to view their contents as well.

Modeling

Training models with Azure Batch AI

The `run_batch_ai.py` script in the "Code\02_Modeling" subfolder of the Workbench project is used to issue a Batch AI Training job. This job retrains an image classifier DNN chosen by the user (AlexNet or ResNet 18 pretrained on

ImageNet). The depth of retraining can also be specified: retraining just the final layer of the network may reduce overfitting when few training samples are available, while fine-tuning the whole network (or, for AlexNet, the fully connected layers) can lead to greater model performance when the training set is sufficiently large.

When the training job completes, this script saves the model (along with a file describing the mapping between the model's integer output and the string labels) and the predictions to blob storage. The BAIT job's log file is parsed to extract the timecourse of error rate improvement over the training epochs. The error rate improvement timecourse is logged to AML Workbench's run history feature for later viewing.

Select a name for your trained model, a pretrained model type, and a retraining depth. Write your selections where indicated in the following command, then begin retraining by executing the command from an Azure ML Command Line Interface:

```
az ml experiment submit -c local Code\02_Modeling\run_batch_ai.py --config_filename Code/settings.cfg --output_model_name [unique model name, alphanumeric characters only] --pretrained_model_type {alexnet,resnet18} --retraining_type {last_only,fully_connected,all} --num_epochs 10
```

Expect the Azure Machine Learning run to take about half an hour to complete. We recommend that you run a few similar commands (varying the output model name, the pretrained model type, and the retraining depth) so that you can compare the performance of models trained with different methods.

Training models with MMLSpark

The `run_mmlspark.py` script in the "Code\02_Modeling" subfolder of the Workbench project is used to train an **MMLSpark** model for image classification. The script first featurizes the training set images using an image classifier DNN pretrained on the ImageNet dataset (either AlexNet or an 18-layer ResNet). The script then uses the featurized images to train an MMLSpark model (either a random forest or a logistic regression model) to classify the images. The test image set is then featurized and scored with the trained model. The accuracy of the model's predictions on the test set is calculated and logged to Azure Machine Learning Workbench's run history feature. Finally, the trained MMLSpark model and its predictions on the test set are saved to blob storage.

Select a unique output model name for your trained model, a pretrained model type, and an MMLSpark model type. Write your selections where indicated in the following command template, then begin retraining by executing the command from an Azure ML Command Line Interface:

```
az ml experiment submit -c myhdi Code\02_Modeling\run_mmlspark.py --config_filename Code/settings.cfg --output_model_name [unique model name, alphanumeric characters only] --pretrained_model_type {alexnet,resnet18} --mmlspark_model_type {randomforest,logisticregression}
```

An additional `--sample_frac` parameter can be used to train and test the model with a subset of available data. Using a small sample fraction decreases runtime and memory requirements at the expense of trained model accuracy. (For example, a run with `--sample_frac 0.1` is expected to take roughly twenty minutes.) For more information on this and other parameters, run `python Code\02_Modeling\run_mmlspark.py -h`.

Users are encouraged to run this script several times with different input parameters. The performance of the resulting models can then be compared in Azure Machine Learning Workbench's Run History feature.

Comparing model performance using the Workbench Run History feature

After you have executed two or more training runs of either type, navigate to the Run History feature in Workbench by clicking the clock icon along the left-hand menu bar. Select `run_mmlspark.py` from the list of scripts at left. A pane loads comparing the test set accuracy for all runs. To see more detail, scroll down and click on the name of an individual run.

Deployment

To apply one of your trained models to aerial images tiling Middlesex County, MA using remote execution on HDInsight, insert your desired model's name into the following command and execute it:

```
az ml experiment submit -c myhdi Code\03_Deployment\batch_score_spark.py --config_filename Code/settings.cfg --output_model_name [trained model name chosen earlier]
```

An additional `--sample_frac` parameter can be used to operationalize the model with a subset of available data.

Using a small sample fraction decreases runtime and memory requirements at the expense of prediction completeness. For more information on this and other parameters, run

```
python Code\03_Deployment\batch_score_spark -h .
```

This script writes the model's predictions to your storage account. The predictions can be examined as described in the next section.

Visualization

The "Model prediction analysis" Jupyter notebook in the "Code\04_Result_Analysis" subfolder of the Workbench project visualizes a model's predictions. Load and run the notebook as follows:

1. Open the project in Workbench and click on the folder ("Files") icon along the left-hand menu to load the directory listing.
2. Navigate to the "Code\04_Result_Analysis" subfolder and click on the notebook named "Model prediction analysis." A preview rendering of the notebook should be displayed.
3. Click "Start Notebook Server" to load the notebook.
4. In the first cell, enter the name of the model whose results you would like to analyze where indicated.
5. Click on "Cell -> Run All" to execute all cells in the notebook.
6. Read along with the notebook to learn more about the analyses and visualizations it presents.

Cleanup

When you have completed the example, we recommend that you delete all of the resources you have created by executing the following command from the Azure Command Line Interface:

```
az group delete --name %AZURE_RESOURCE_GROUP%
```

References

- [The Embarrassingly Parallel Image Classification repository](#)
 - Describes dataset construction from freely available imagery and labels
- [MMLSpark GitHub repository](#)
 - Contains additional examples of model training and evaluation with MMLSpark

Conclusions

Azure Machine Learning Workbench helps data scientists easily deploy their code on remote compute targets. In this example, local MMLSpark training code was deployed for remote execution on an HDInsight cluster, and a local script launched a training job on an Azure Batch AI GPU cluster. Azure Machine Learning Workbench's run history feature tracked the performance of multiple models and helped us identify the most accurate model. Workbench's Jupyter notebooks feature helped visualize our models' predictions in an interactive, graphical environment.

Next steps

To dive deeper into this example:

- In Azure Machine Learning Workbench's Run History feature, click the gear symbols to select which graphs and metrics are displayed.
- Examine the sample scripts for statements calling the `run_logger`. Check that you understand how each metric is being recorded.
- Examine the sample scripts for statements calling the `blob_service`. Check that you understand how trained models and predictions are stored and retrieved from the cloud.
- Explore the contents of the containers created in your blob storage account. Ensure that you understand which script or command is responsible for creating each group of files.
- Modify the training script to train a different MMLSpark model type or to change the model hyperparameters. Use the run history feature to determine whether your changes increased or decreased the model's accuracy.

Server workload forecasting on terabytes of data

10/30/2018 • 19 minutes to read • [Edit Online](#)

NOTE

This article is deprecated. Support for earlier versions of this service will end incrementally. [View the support timeline](#). Start using the latest version with this [quickstart](#).

This article covers how data scientists can use Azure Machine Learning Workbench to develop solutions that require the use of big data. You can start from a sample of a large data set, iterate through data preparation, feature engineering, and machine learning, and then extend the process to the entire large data set.

You'll learn about the following key capabilities of Machine Learning Workbench:

- Easy switching between compute targets. You can set up different compute targets and use them in experimentation. In this example, you use an Ubuntu DSVM and an Azure HDInsight cluster as the compute targets. You can also configure the compute targets, depending on the availability of resources. In particular, after scaling out the Spark cluster with more worker nodes, you can use the resources through Machine Learning Workbench to speed up experiment runs.
- Run history tracking. You can use Machine Learning Workbench to track the performance of machine learning models and other metrics of interest.
- Operationalization of the machine learning model. You can use the built-in tools within Machine Learning Workbench to deploy the trained machine learning model as a web service on Azure Container Service. You can also use the web service to get mini-batch predictions through REST API calls.
- Support for terabytes data.

NOTE

For code samples and other materials related to this example, see [GitHub](#).

Use case overview

Forecasting the workload on servers is a common business need for technology companies that manage their own infrastructure. To reduce infrastructure cost, services running on under-used servers should be grouped together to run on a smaller number of machines. Services running on overused servers should be given more machines to run.

In this scenario, you focus on workload prediction for each machine (or server). In particular, you use the session data on each server to predict the workload class of the server in future. You classify the load of each server into low, medium, and high classes by using the Random Forest Classifier in [Apache Spark ML](#). The machine learning techniques and workflow in this example can be easily extended to other similar problems.

Prerequisites

The prerequisites to run this example are as follows:

- An [Azure account](#) (free trials are available).
- An installed copy of [Azure Machine Learning Workbench](#). To install the program and create a workspace, see the [quickstart installation guide](#). If you have multiple subscriptions, you can [set the desired subscription to be](#)

the current active subscription.

- Windows 10 (the instructions in this example are generally the same for macOS systems).
- A Data Science Virtual Machine (DSVM) for Linux (Ubuntu), preferably in East US region where the data locates. You can provision an Ubuntu DSVM by following [these instructions](#). You can also see [this quickstart](#). We recommend using a virtual machine with at least 8 cores and 32 GB of memory.

Follow the [instruction](#) to enable password-less sudoer access on the VM for AML Workbench. You can choose to use [SSH key-based authentication for creating and using the VM in AML Workbench](#). In this example, we use password to access the VM. Save the following table with the DSVM info for later steps:

FIELD NAME	VALUE
DSVM IP address	xxx
User name	xxx
Password	xxx

You can choose to use any VM with [Docker Engine](#) installed.

- An HDInsight Spark Cluster, with Hortonworks Data Platform version 3.6 and Spark version 2.1.x, preferably in East US region where the data locates. Visit [Create an Apache Spark cluster in Azure HDInsight](#) for details about how to create HDInsight clusters. We recommend using a three-worker cluster, with each worker having 16 cores and 112 GB of memory. Or you can just choose VM type `D12 v2` for head node, and `D14 v2` for the worker node. The deployment of the cluster takes about 20 minutes. You need the cluster name, SSH user name, and password to try out this example. Save the following table with the Azure HDInsight cluster info for later steps:

FIELD NAME	VALUE
Cluster name	xxx
User name	xxx (sshuser by default)
Password	xxx

- An Azure Storage account. You can follow [these instructions](#) to create one. Also, create two private blob containers with the names `fullmodel` and `onemonthmodel` in this storage account. The storage account is used to save intermediate compute results and machine learning models. You need the storage account name and access key to try out this example. Save the following table with the Azure storage account info for later steps:

FIELD NAME	VALUE
Storage account name	xxx
Access key	xxx

The Ubuntu DSVM and the Azure HDInsight cluster created in the prerequisite list are compute targets. Compute targets are the compute resource in the context of Machine Learning Workbench, which might be different from the computer where Workbench runs.

Create a new Workbench project

Create a new project by using this example as a template:

1. Open Machine Learning Workbench.
2. On the **Projects** page, select the + sign, and select **New Project**.
3. In the **Create New Project** pane, fill in the information for your new project.
4. In the **Search Project Templates** search box, type **Workload Forecasting on Terabytes Data**, and select the template.
5. Select **Create**.

You can create a Workbench project with a pre-created git repository by following [this instruction](#).

Run `git status` to inspect the status of the files for version tracking.

Data description

The data used in this example is synthesized server workload data. It is hosted in an Azure Blob storage account that's publicly accessible in East US region. The specific storage account info can be found in the `dataFile` field of [Config/storageconfig.json](#) in the format of "wasb://@.blob.core.windows.net/". You can use the data directly from the Blob storage. If the storage is used by many users simultaneously, you can use [azcopy](#) to download the data into your own storage for better experimentation experience.

The total data size is approximately 1 TB. Each file is about 1-3 GB, and is in CSV file format, without header. Each row of data represents the load of a transaction on a particular server. The detailed information of the data schema is as follows:

COLUMN NUMBER	FIELD NAME	TYPE	DESCRIPTION
1	<code>SessionStart</code>	Datetime	Session start time
2	<code>SessionEnd</code>	Datetime	Session end time
3	<code>ConcurrentConnectionCounts</code>	Integer	Number of concurrent connections
4	<code>MbytesTransferred</code>	Double	Normalized data transferred in megabytes
5	<code>ServiceGrade</code>	Integer	Service grade for the session
6	<code>HTTP1</code>	Integer	Session uses HTTP1 or HTTP2
7	<code>ServerType</code>	Integer	Server type
8	<code>SubService_1_Load</code>	Double	Subservice 1 load
9	<code>SubService_2_Load</code>	Double	Subservice 2 load
10	<code>SubService_3_Load</code>	Double	Subservice 3 load
11	<code>SubService_4_Load</code>	Double	Subservice 4 load
12	<code>SubService_5_Load</code>	Double	Subservice 5 load

COLUMN NUMBER	FIELD NAME	TYPE	DESCRIPTION
13	SecureBytes_Load	Double	Secure bytes load
14	TotalLoad	Double	Total load on server
15	ClientIP	String	Client IP address
16	ServerIP	String	Server IP address

Note that the expected data types are listed in the preceding table. Due to missing values and dirty-data problems, there is no guarantee that the data types actually are as expected. Data processing should take this into consideration.

Scenario structure

The files in this example are organized as follows.

FILE NAME	TYPE	DESCRIPTION
Code	Folder	The folder contains all the code in the example.
Config	Folder	The folder contains the configuration files.
Image	Folder	The folder used to save images for the README file.
Model	Folder	The folder used to save model files downloaded from Blob storage.
Code/etl.py	Python file	The Python file used for data preparation and feature engineering.
Code/train.py	Python file	The Python file used to train a three-class multi-classification model.
Code/webservice.py	Python file	The Python file used for operationalization.
Code/scoring_webservice.py	Python file	The Python file used for data transformation and calling the web service.
Code/016Npreprocessing.py	Python file	The Python file used to preprocess the data for scoring_webservice.py.
Code/util.py	Python file	The Python file that contains code for reading and writing Azure blobs.
Config/storageconfig.json	JSON file	The configuration file for the Azure blob container that stores the intermediate results and model for processing and training on one-month data.

FILE NAME	TYPE	DESCRIPTION
Config/fulldata_storageconfig.json	Json file	The configuration file for the Azure blob container that stores the intermediate results and model for processing and training on a full data set.
Config/webservice.json	JSON file	The configuration file for scoring_webservice.py.
Config/conda_dependencies.yml	YAML file	The Conda dependency file.
Config/conda_dependencies_webservice.yml	YAML file	The Conda dependency file for the web service.
Config/dsvm_spark_dependencies.yml	YAML file	The Spark dependency file for Ubuntu DSVM.
Config/hdi_spark_dependencies.yml	YAML file	The Spark dependency file for the HDInsight Spark cluster.
README.md	Markdown file	The README markdown file.
Code/download_model.py	Python file	The Python file used to download the model files from the Azure blob to a local disk.
Docs/DownloadModelsFromBlob.md	Markdown file	The markdown file that contains instructions for how to run <code>Code/download_model.py</code> .

Data flow

The code in `Code/etl.py` loads data from the publicly accessible container (`dataFile` field of `Config/storageconfig.json`). It includes data preparation and feature engineering, and saves the intermediate compute results and models to your own private container. The code in `Code/train.py` loads the intermediate compute results from the private container, trains the multi-class classification model, and writes the trained machine learning model to the private container.

You should use one container for experimentation on the one-month data set, and another one for experimentation on the full data set. Because the data and models are saved as Parquet file, each file is actually a folder in the container, containing multiple blobs. The resulting container looks as follows:

BLOB PREFIX NAME	TYPE	DESCRIPTION
featureScaleModel	Parquet	Standard scaler model for numeric features.
stringIndexModel	Parquet	String indexer model for non-numeric features.
oneHotEncoderModel	Parquet	One-hot encoder model for categorical features.
mlModel	Parquet	Trained machine learning model.

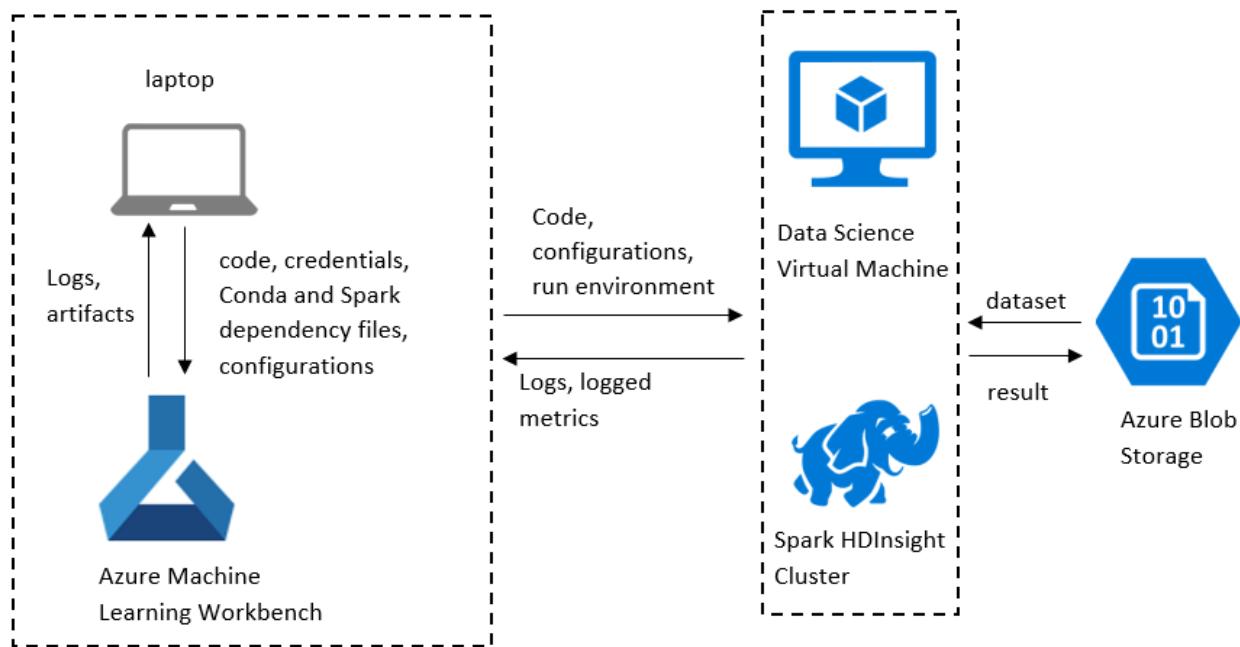
BLOB PREFIX NAME	TYPE	DESCRIPTION
info	Python pickle file	Information about the transformed data, including training start, training end, duration, the timestamp for train-test splitting, and columns for indexing and one-hot encoding.

All the files and blobs in the preceding table are used for operationalization.

Model development

Architecture diagram

The following diagram shows the end-to-end workflow of using Machine Learning Workbench to develop the model:



In the following sections, we show the model development by using the remote compute target functionality in Machine Learning Workbench. We first load a small amount of sample data, and run the script [Code/etl.py](#) on an Ubuntu DSVM for fast iteration. We can further limit the work we do in [Code/etl.py](#) by passing an extra argument for faster iteration. In the end, we use an HDInsight cluster to train with full data.

The [Code/etl.py](#) file loads and prepares the data, and performs feature engineering. It accepts two arguments:

- A configuration file for the Blob storage container, for storing the intermediate compute results and models.
- A debug config argument for faster iteration.

The first argument, `configFilename`, is a local configuration file where you store the Blob storage information, and specify where to load the data. By default, it is [Config/storageconfig.json](#), and it is going to be used in the one-month data run. We also include [Config/fulldata_storageconfig.json](#), which you need to use on the full data set run. The content in the configuration is as follows:

FIELD	TYPE	DESCRIPTION
storageAccount	String	Azure Storage account name
storageContainer	String	Container in Azure Storage account to store intermediate results

FIELD	TYPE	DESCRIPTION
storageKey	String	Azure Storage account access key
dataFile	String	Data source files
duration	String	Duration of data in the data source files

Modify both `Config/storageconfig.json` and `Config/fulldata_storageconfig.json` to configure the storage account, storage key, and the blob container to store the intermediate results. By default, the blob container for the one-month data run is `onemonthmodel`, and the blob container for full data set run is `fullmodel`. Make sure you create these two containers in your storage account. The `dataFile` field in `Config/fulldata_storageconfig.json` configures what data is loaded in `Code/etl.py`. The `duration` field configures the range the data includes. If the duration is set to ONE_MONTH, the data loaded should be just one .csv file among the seven files of the data for June-2016. If the duration is FULL, the full data set (1 TB) is loaded. You don't need to change `dataFile` and `duration` in these two configuration files.

The second argument is DEBUG. Setting it to FILTER_IP enables a faster iteration. Use of this parameter is helpful when you want to debug your script.

NOTE

In all of the following commands, replace any argument variable with its actual value.

Model development on the Docker of Ubuntu DSVM

1. Set up the compute target

Start the command line from Machine Learning Workbench by selecting **File > Open Command Prompt**. Then run:

```
az ml computetarget attach remotedocker --name dockerdsvm --address $DSVMIPAddress --username $user --password $password
```

The following two files are created in the aml_config folder of your project:

- `dockerdsvm.compute`: This file contains the connection and configuration information for a remote execution target.
- `dockerdsvm.runconfig`: This file is a set of run options used within the Workbench application.

Browse to `dockerdsvm.runconfig`, and change the configuration of these fields to the following:

```
PrepareEnvironment: true
CondaDependenciesFile: Config/conda_dependencies.yml
SparkDependenciesFile: Config/dsvm_spark_dependencies.yml
```

Prepare the project environment by running:

```
az ml experiment prepare -c dockerdsvm
```

With `PrepareEnvironment` set to true, Machine Learning Workbench creates the runtime environment whenever you submit a job. `Config/conda_dependencies.yml` and `Config/dsvm_spark_dependencies.yml` contain the customization of the runtime environment. You can always modify the Conda dependencies, Spark configuration, and Spark dependencies by editing these two YMAL files. For this example, we added `azure-storage` and `azure-ml-api-sdk` as extra Python packages in `Config/conda_dependencies.yml`. We also added `spark.default.parallelism`, `spark.executor.instances`, and `spark.executor.cores` in `Config/dsvm_spark_dependencies.yml`.

2. Data preparation and feature engineering on DSVM Docker

Run the script `etl.py` on DSVM Docker. Use a debug parameter that filters the loaded data with specific server IP addresses:

```
az ml experiment submit -t dockerdsvm -c dockerdsvm ./Code/etl.py ./Config/storageconfig.json FILTER_IP
```

Browse to the side panel, and select **Run** to see the run history of `etl.py`. Notice that the run time is about two minutes. If you plan to change your code to include new features, providing `FILTER_IP` as the second argument provides a faster iteration. You might need to run this step multiple times when dealing with your own machine learning problems, to explore the data set or create new features.

With the customized restriction on what data to load, and further filtering of what data to process, you can speed up the iteration process in your model development. As you experiment, you should periodically save the changes in your code to the Git repository. Note that we used the following code in `etl.py` to enable the access to the private container:

```
def attach_storage_container(spark, account, key):
    config = spark._sc._jsc.hadoopConfiguration()
    setting = "fs.azure.account.key." + account + ".blob.core.windows.net"
    if not config.get(setting):
        config.set(setting, key)

# attach the blob storage to the spark cluster or VM so that the storage can be accessed by the cluster or VM
attach_storage_container(spark, storageAccount, storageKey)
```

Next, run the script `etl.py` on DSVM Docker without the debug parameter `FILTER_IP`:

```
az ml experiment submit -t dockerdsvm -c dockerdsvm ./Code/etl.py ./Config/storageconfig.json FALSE
```

Browse to the side panel, and select **Run** to see the run history of `etl.py`. Notice that the run time is about four minutes. The processed result of this step is saved into the container, and is loaded for training in `train.py`. In addition, the string indexers, encoder pipelines, and standard scalers are saved to the private container. These are used in operationalization.

3. Model training on DSVM Docker

Run the script `train.py` on DSVM Docker:

```
az ml experiment submit -t dockerdsvm -c dockerdsvm ./Code/train.py ./Config/storageconfig.json
```

This step loads the intermediate compute results from the run of `etl.py`, and trains a machine learning model. This step takes about two minutes.

When you have successfully finished the experimentation on the small data, you can continue to run the experimentation on the full data set. You can start by using the same code, and then experiment with argument and compute target changes.

Model development on the HDInsight cluster

1. Create the compute target in Machine Learning Workbench for the HDInsight cluster

```
az ml computetarget attach cluster --name myhdi --address $clustername-ssh.azurehdinsight.net --username
$username --password $password
```

The following two files are created in the `aml_config` folder:

- `myhdi.compute`: This file contains connection and configuration information for a remote execution target.
- `myhdi.runconfig`: This file is set of run options used within the Workbench application.

Browse to `myhdi.runconfig`, and change the configuration of these fields to the following:

```
PrepareEnvironment: true  
CondaDependenciesFile: Config/conda_dependencies.yml  
SparkDependenciesFile: Config/hdi_spark_dependencies.yml
```

Prepare the project environment by running:

```
az ml experiment prepare -c myhdi
```

This step can take up to seven minutes.

2. Data preparation and feature engineering on HDInsight cluster

Run the script `etl.py`, with full data on the HDInsight cluster:

```
az ml experiment submit -a -t myhdi -c myhdi ./Code/etl.py Config/fulldata_storageconfig.json FALSE
```

Because this job lasts for a relatively long time (approximately two hours), you can use `-a` to disable output streaming. When the job is done, in **Run History**, you can view the driver and controller logs. If you have a larger cluster, you can always reconfigure the configurations in `Config/hdi_spark_dependencies.yml` to use more instances or cores. For example, if you have a four-worker-node cluster, you can increase the value of `spark.executor.instances` from 5 to 7. You can see the output of this step in the **fullmodel** container in your storage account.

3. Model training on HDInsight cluster

Run the script `train.py` on HDInsight cluster:

```
az ml experiment submit -a -t myhdi -c myhdi ./Code/train.py Config/fulldata_storageconfig.json
```

Because this job lasts for a relatively long time (approximately 30 minutes), you can use `-a` to disable output streaming.

Run history exploration

Run history is a feature that enables tracking of your experimentation in Machine Learning Workbench. By default, it tracks the duration of the experimentation. In our specific example, when we move to the full data set for `Code/etl.py` in the experimentation, we notice that duration significantly increases. You can also log specific metrics for tracking purposes. To enable metric tracking, add the following lines of code to the head of your Python file:

```
# import logger  
from azureml.logging import get_azureml_logger  
  
# initialize logger  
run_logger = get_azureml_logger()
```

Here is an example to track a specific metric:

```
run_logger.log("Test Accuracy", testAccuracy)
```

On the right sidebar of the Workbench, browse to **Runs** to see the run history for each Python file. You can also go to your GitHub repository. A new branch, with the name starting with "AMLHistory," is created to track the changes you made to your script in each run.

Operationalize the model

In this section, you operationalize the model you created in the previous steps as a web service. You also learn how to use the web service to predict workload. Use Machine Learning operationalization command-line interfaces (CLIs) to package the code and dependencies as Docker images, and to publish the model as a containerized web service.

You can use the command-line prompt in Machine Learning Workbench to run the CLIs. You can also run the CLIs on Ubuntu Linux by following the [installation guide](#).

NOTE

In all the following commands, replace any argument variable with its actual value. It takes about 40 minutes to finish this section.

Choose a unique string as the environment for operationalization. Here, we use the string "[unique]" to represent the string you choose.

1. Create the environment for operationalization, and create the resource group.

```
az ml env setup -c -n [unique] --location eastus2 --cluster -z 5 --yes
```

Note that you can use Container Service as the environment by using `--cluster` in the `az ml env setup` command. You can operationalize the machine learning model on [Azure Container Service](#). It uses [Kubernetes](#) for automating deployment, scaling, and management of containerized applications. This command takes about 20 minutes to run. Use the following to check if the deployment has finished successfully:

```
az ml env show -g [unique]rg -n [unique]
```

Set the deployment environment as the one you just created by running the following:

```
az ml env set -g [unique]rg -n [unique]
```

2. Create and use a model management account. To create a model management account, run the following:

```
az ml account modelmanagement create --location eastus2 -n [unique]acc -g [unique]rg --sku-instances 4  
--sku-name S3
```

Use the model management for operationalization by running the following:

```
az ml account modelmanagement set -n [unique]acc -g [unique]rg
```

A model management account is used to manage the models and web services. From the Azure portal, you can see a new model management account has been created. You can see the models, manifests, Docker images, and services that are created by using this model management account.

3. Download and register the models.

Download the models in the **fullmodel** container to your local machine in the directory of code. Do not download the parquet data file with the name "vmlSource.parquet." It's not a model file; it's an intermediate compute result. You can also reuse the model files included in the Git repository. For more information, see [GitHub](#).

Go to the `Model` folder in the CLI, and register the models as follows:

```
az ml model register -m mlModel -n vmlModel -t fullmodel  
az ml model register -m featureScaleModel -n featureScaleModel -t fullmodel  
az ml model register -m oneHotEncoderModel -n oneHotEncoderModel -t fullmodel  
az ml model register -m stringIndexModel -n stringIndexModel -t fullmodel  
az ml model register -m info -n info -t fullmodel
```

The output of each command gives a model ID, which is needed in the next step. Save them in a text file for future use.

4. Create a manifest for the web service.

A manifest includes the code for the web service, all the machine learning models, and runtime environment dependencies. Go to the `Code` folder in the CLI, and run the following command:

```
az ml manifest create -n $webserviceName -f webservice.py -r spark-py -c  
..../Config/conda_dependencies_webservice.yml -i $modelID1 -i $modelID2 -i $modelID3 -i $modelID4 -i  
$modelID5
```

The output gives a manifest ID for the next step.

Stay in the `Code` directory, and you can test `webservice.py` by running the following:

```
az ml experiment submit -t dockerdsvm -c dockerdsvm webservice.py
```

5. Create a Docker image.

```
az ml image create -n [unique]image --manifest-id $manifestID
```

The output gives an image ID for the next step. This docker image is used in Container Service.

6. Deploy the web service to the Container Service cluster.

```
az ml service create realtime -n [unique] --image-id $imageID --cpu 0.5 --memory 2G
```

The output gives a service ID. You need to use it to get the authorization key and service URL.

7. Call the web service in Python code to score in mini-batches.

Use the following command to get the authorization key:

```
az ml service keys realtime -i $ServiceID
```

Use the following command to get the service scoring URL:

```
az ml service usage realtime -i $ServiceID
```

Modify the content in `./Config/webservice.json` with the right service scoring URL and authorization key. Keep the "Bearer" in the original file, and replace the "xxx" part.

Go to the root directory of your project, and test the web service for mini-batch scoring by using the following:

```
az ml experiment submit -t dockerdsvm -c dockerdsvm ./Code/scoring_webservice.py  
./Config/webservice.json
```

8. Scale the web service.

For more information, see [How to scale operationalization on your Azure Container Service cluster](#).

Next steps

This example highlights how to use Machine Learning Workbench to train a machine learning model on big data, and operationalize the trained model. In particular, you learned how to configure and use different compute targets, and run the history of tracking metrics and use different runs.

You can extend the code to explore cross-validation and hyper-parameter tuning. To learn more about cross-validation and hyper-parameter tuning, see [this GitHub resource](#).

To learn more about time-series forecasting, see [this GitHub resource](#).

Energy Demand Time Series Forecasting

9/24/2018 • 12 minutes to read • [Edit Online](#)

NOTE

This article is deprecated. Support for earlier versions of this service will end incrementally. [View the support timeline](#). Start using the latest version with this [quickstart](#).

Time series forecasting is the task of predicting future values in a time-ordered sequence of observations. It is a common problem and has applications in many industries. For example, retail companies need to forecast future product sales so they can effectively organize their supply chains to meet demand. Similarly, package delivery companies need to estimate the demand for their services so they can plan workforce requirements and delivery routes ahead of time. In many cases, the financial risks of inaccurate forecasts can be significant. Therefore, forecasting is often a business critical activity.

This sample shows how time series forecasting can be performed through applying machine learning techniques. You are guided through every step of the modeling process including:

- Data preparation to clean and format the data;
- Creating features for the machine learning models from raw time series data;
- Training various machine learning models;
- Evaluating the models by comparing their performance on a held-out test dataset; and,
- Operationalizing the best model, making it available through a web service to generate forecasts on demand.

Azure Machine Learning Workbench aids the modeling process at every step:

- The sample shows how the Jupyter notebook environment - available directly from through Workbench - can make developing Python code easier. The model development process can be explained to others more clearly using markdown annotations and charts. These notebooks can be viewed, edited, and executed directly from the Workbench.
- Trained models can be persisted and uploaded to blob storage. This helps the data scientist to keep track of trained model objects and ensure they are retained and retrievable when needed.
- Metrics can be logged while executing a Python script, enabling the data scientist to keep a record of model performance scores.
- The workbench produces customizable tables of logged metrics allowing the data scientist to easily compare model performance metrics. Charts are automatically displayed so the best performing model can be readily identified.
- Finally, the sample shows how a trained model can be operationalized by deploying it in a realtime web service.

Link to the Gallery GitHub repository

The public GitHub repository for this real world scenario contains all materials, including code samples, needed for this example:

<https://github.com/Azure/MachineLearningSamples-EnergyDemandTimeSeriesForecasting>

Use case overview

This scenario focuses on energy demand forecasting where the goal is to predict the future load on an energy grid.

It is a critical business operation for companies in the energy sector as operators need to maintain the fine balance between the energy consumed on a grid and the energy supplied to it. Too much power supplied to the grid can result in waste of energy or technical faults. However, if too little power is supplied it can lead to blackouts, leaving customers without power. Typically, grid operators can take short-term decisions to manage energy supply to the grid and keep the load in balance. An accurate short-term forecast of energy demand is therefore essential for the operator to make these decisions with confidence.

This scenario details the construction of a machine learning energy demand forecasting solution. The solution is trained on a public dataset from the [New York Independent System Operator \(NYISO\)](#), which operates the power grid for New York State. The dataset includes hourly power demand data for New York City over a period of five years. An additional dataset containing hourly weather conditions in New York City over the same time period was taken from [darksky.net](#).

Prerequisites

- An [Azure account](#) (free trials are available).
- An installed copy of [Azure Machine Learning Workbench](#) following the [quick start installation guide](#) to install the program and create a workspace.
- This sample assumes that you are running Azure ML Workbench on Windows 10 with [Docker engine](#) locally installed. If you are using macOS, the instructions are largely the same.
- Azure Machine Learning Operationalization installed with a local deployment environment set up and a model management account created as described in this [guide](#).
- This sample requires that you update the Pandas installation to version 0.20.3 or higher and install matplotlib. Click *Open Command Prompt* from the *File* menu in the Workbench and run the following commands to install these dependencies:

```
conda install "pandas>=0.21.1"
```

Create a new Workbench project

Create a new project using this example as a template:

1. Open Azure Machine Learning Workbench
2. On the **Projects** page, click the + sign and select **New Project**
3. In the **Create New Project** pane, fill in the information for your new project
4. In the **Search Project Templates** search box, type "Energy Demand Time Series Forecasting" and select the template
5. Click **Create**

Data description

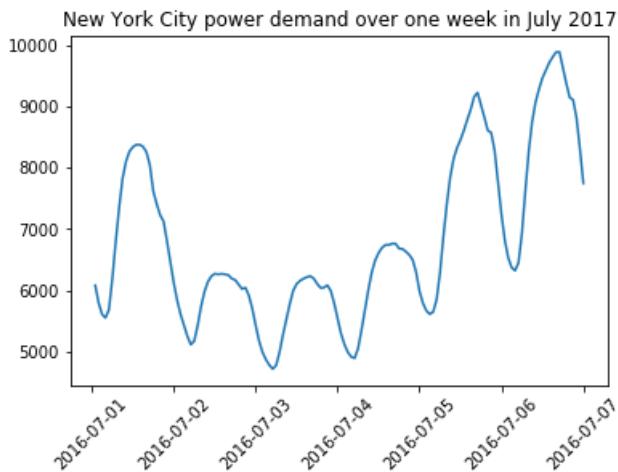
Two datasets are provided with this sample and are downloaded using the [1-data-preparation.ipynb](#) notebook: [nyc_demand.csv](#) and [nyc_weather.csv](#).

nyc_demand.csv contains hourly energy demand values for New York City for the years 2012-2017. The data has the following simple structure:

TIMESTAMP	DEMAND
2012-01-01 00:00:00	4937.5
2012-01-01 01:00:00	4752.1

TIMESTAMP	DEMAND
2012-01-01 02:00:00	4542.6
2012-01-01 03:00:00	4357.7

Demand values are in megawatt-hours (MWh). Below is a chart of energy demand over a 7-day period in July 2017:



nyc_weather.csv contains hourly weather values for New York City over the years 2012-2017:

TIMESTAMP	PRECIP	TEMP
2012-01-01 00:00:00	0.0	46.13
2012-01-01 01:00:00	0.01	45.89
2012-01-01 02:00:00	0.05	45.04
2012-01-01 03:00:00	0.02	45.03

precip is a percentage measure of the level of precipitation. *temp* (temperature) values have been rescaled such that all values fall in the interval [0, 100].

Scenario structure

When you open this project for the first time in Azure Machine Learning Workbench, navigate to the *Files* pane on the left-hand side. The following code files are provided with this sample:

- `1-data-preparation.ipynb` - this Jupyter notebook downloads and processes the data to prepare it for modeling. This is the first notebook you will run.
- `2-linear-regression.ipynb` - this notebook trains a linear regression model on the training data.
- `3-ridge.ipynb` - trains a ridge regression model.
- `4-ridge-poly2.ipynb` - trains a ridge regression model on polynomial features of degree 2.
- `5-mlp.ipynb` - trains a multi-layer perceptron neural network.
- `6-dtree.ipynb` - trains a decision tree model.
- `7-gbm.ipynb` - trains a gradient boosted machine model.
- `8-evaluate-model.py` - this script loads a trained model and uses it to make predictions on a held-out test dataset. It produces model evaluation metrics so the performance of different models can be compared.

- `9-forecast-output-exploration.ipynb` - this notebook produces visualizations of the forecasts generated by the machine learning models.
- `10-deploy-model.ipynb` - this notebook demonstrates how a trained forecasting model can be operationalized in a realtime web service.
- `evaluate-all-models.py` - this script evaluates all trained models. It provides an alternative to running `8-evaluate-model.py` for each trained model individually.

1. Data preparation and cleaning

To start running the sample, first click on `1-data-preparation.ipynb` to open the notebook in preview mode. Click on **Start Notebook Server** to start a Jupyter notebook server and Python kernel on your machine. You can either run the notebooks from within the Workbench, or you can use your browser by navigating to <http://localhost:8888>. Note that you must change the kernel to *PROJECT_NAME local*. You can do this from the *Kernel* menu in the notebook under *Change kernel*. Press **shift+Enter** to run the code in individual notebook cells, or click *Run All* in the *Cell* menu to run the entire notebook.

The notebook first downloads the data from a blob storage container hosted on Azure. The data is then stored on your machine in the `AZUREML_NATIVE_SHARE_DIRECTORY`. This location is accessible from any notebooks or scripts you run from the Workbench so is a good place to store data and trained models.

Once the data has been downloaded, the notebook cleans the data by filling gaps in the time series and filling missing values by interpolation. Cleaning the time series data in this way maximizes the amount of data available for training the models.

Several model features are created from the cleaned raw data. These features can be categorized into two groups:

- **Time driven features** are derived from the *timestamp* variable e.g. *month*, *dayofweek* and *hour*.
- **Lagged features** are time shifted values of the actual demand or temperature values. These features aim to capture the conditional dependencies between consecutive time periods in the model.

The resulting feature dataset can then be used when developing forecasting models.

2. Model development

A first modeling approach may be a simple linear regression model. The `2-linear-regression.ipynb` notebook shows how to train a linear regression forecast model for future energy demand. In particular, the model will be trained to predict energy demand one hour ($t+1$) ahead of the current time period (t). However, we can apply this 'one-step' model recursively at test time to generate forecasts for future time periods, $t+n$.

To train a model, a predictive pipeline is created which involves three distinct steps:

- **A data transformation:** the required formats for input data can vary depending on the machine learning algorithm. In this case, the linear regression model requires categorical variables to be one-hot encoded.
- **A cross validation routine:** Often a machine learning model will have one or more hyperparameters that need to be tuned through experimentation. Cross validation can be used to find the optimal set of parameter values. The model is repeatedly trained and evaluated on different folds of the dataset. The best parameters are those that achieve the best model performance when averaged across the cross validation folds. This process is explained in more detail in `2-linear-regression.ipynb`.
- **Training the final model:** The model is trained on the whole dataset, using the best set of hyperparameters.

We have included a series of 6 different models in notebooks numbered 2-7. Each notebook trains a different model and stores it in the `AZUREML_NATIVE_SHARE_DIRECTORY` location. Once you have trained all the models developed for this scenario, we can evaluate and compare them as described in the next section.

3. Model evaluation and comparison

We can use a trained model to make forecasts for future time periods. It is best to evaluate these models on a held-out test dataset. By evaluating the different models on the same unseen dataset, we can fairly compare their

performance and identify the best model. In this scenario, we apply the trained model recursively to forecast multiple time steps into the future. In particular, we generate forecasts for up to six hours, $t+6$ ahead of the current hour, t . These predictions are evaluated against the actual demand observed for each time period.

To evaluate a model, open the `8-evaluate-model.py` script from the *Files* pane in the Workbench. Check that *Run Configuration* is set to **local** and then type the model name into the *Arguments* field. The model name needs to match exactly the *model_name* variable set at the start of the notebook in which the model is trained. For example, enter "linear_regression" to evaluate the trained linear regression model. Alternatively, once all models have been trained, you can evaluate them all with one command by running `evaluate-all-models.py`. To run this script, open a command prompt from the Workbench and execute the following command:

```
python evaluate-all-models.py
```

The `8-evaluate-model.py` script performs the following operations:

- Loads a trained model pipeline from disk
- Makes predictions on the test dataset
- Computes model performance metrics and logs them
- Saves the test dataset predictions to `AZUREML_NATIVE_SHARE_DIRECTORY` for later inspection and analysis
- Saves the trained model pipeline to the *outputs* folder.

NOTE

Saving the model to the *outputs* folder automatically copies the model object into the Azure Blob Storage account associated with your Experimentation account. This means you will always be able to retrieve the saved model object from the blob, even if you change machine or compute context.

Navigate to the *Run History* pane and click on `8-evaluate-model.py`. You will see charts displaying several model performance metrics:

- **ME**: mean error of the forecast. This can be interpreted as the average bias of the forecast.
- **MPE**: **mean percentage error** (ME expressed as a percentage of the actual demand)
- **MSE**: **mean squared error**
- **RMSE**: root mean squared error (the square root of MSE)
- **MAPE**: **mean absolute percentage error**
- **sMAPE**: **symmetric mean absolute percentage error**
- **MAPE_base**: MAPE of a baseline forecast in which the prediction equals the last known demand value.
- **MdRAE**: Median Relative Absolute Error. The median ratio of the forecast error to the baseline forecast error. A value less than 1 means the forecast is performing better than the baseline.

All metrics above refer to the $t+1$ forecast. In addition to the above metrics, you will also see a set of corresponding metrics with the *_horizon* suffix. This is the metric averaged over all periods in the forecast range from period $t+1$ to period $t+6$.

If metrics are not displaying in the chart area, click on the gear symbol in the top right-hand corner. Ensure that the model performance metrics you are interested in are checked. The metrics will also appear in a table below the charts. Again, this table is customizable by clicking on the gear symbol. Sort the table by a performance metric to identify the best model. If you are interested in seeing how the forecast performance varies from period $t+1$ to $t+6$, click on the entry for the model in the table. Charts displaying the MAPE, MPE and MdRAE metrics across the forecast period are shown under *Visualizations*.

When evaluating forecasting models, it can be very useful to visualize the output predictions. This helps the data scientist to determine whether the forecast produced appears realistic. It can also help to identify problems in the

forecast if, for example, the forecast performs poorly in certain time periods. The `9-forecast-output-exploration.ipynb` notebook will produce visualizations of the forecasts generated for the test dataset.

4. Deployment

The best model can be operationalized by deploying it as a realtime web service. This web service can then be invoked to generate forecasts on demand as new data becomes available. In this scenario, a new forecast needs to be generated every hour to predict the energy demand in the subsequent hour. To perform this task, a web service can be deployed which takes an array of features for a given hour time period as input and returns the predicted demand as output.

In this sample, a web service is deployed to a Windows 10 machine. Ensure you have completed the prerequisite steps for local deployment set out in this [getting started guide](#) for the Operationalization CLI. Once you have set up your local environment and model management account, run the `10-deploy-model.ipynb` notebook to deploy the web service.

Conclusion

This sample demonstrates how to build an end-to-end time series forecasting solution for the purposes of forecasting energy demand. Many of the principles explored in this sample can be extended to other forecasting scenarios and industries.

This scenario shows how Azure Machine Learning Workbench can assist a data scientist in developing real world solutions with useful features such as the Jupyter notebook environment and metric logging capabilities. The sample also guides the reader on how a model can be operationalized and deployed using Azure Machine Learning Operationalization CLI. Once deployed, the web service API allows developers or data engineers to integrate the forecasting model into a wider data pipeline.

Distributed tuning of hyperparameters using Azure Machine Learning Workbench

9/24/2018 • 14 minutes to read • [Edit Online](#)

NOTE

This article is deprecated. Support for earlier versions of this service will end incrementally. [View the support timeline](#). Start using the latest version with this [quickstart](#).

This scenario shows how to use Azure Machine Learning Workbench to scale out tuning of hyperparameters of machine learning algorithms that implement scikit-learn API. We show how to configure and use a remote Docker container and Spark cluster as an execution backend for tuning hyperparameters.

Link of the Gallery GitHub repository

Following is the link to the public GitHub repository:

<https://github.com/Azure/MachineLearningSamples-DistributedHyperParameterTuning>

Use case overview

Many machine learning algorithms have one or more knobs, called hyperparameters. These knobs allow tuning of algorithms to optimize their performance over future data, measured according to user-specified metrics (for example, accuracy, AUC, RMSE). Data scientist needs to provide values of hyperparameters when building a model over training data and before seeing the future test data. How based on the known training data can we set up the values of hyperparameters so that the model has a good performance over the unknown test data?

A popular technique for tuning hyperparameters is a *grid search* combined with *cross-validation*. Cross-validation is a technique that assesses how well a model, trained on a training set, predicts over the test set. Using this technique, we first divide the dataset into K folds and then train the algorithm K times in a round-robin fashion. We do this on all but one of the folds called the "held-out fold". We compute the average value of the metrics of K models over K held-out folds. This average value, called *cross-validated performance estimate*, depends on the values of hyperparameters used when creating K models. When tuning hyperparameters, we search through the space of candidate hyperparameter values to find the ones that optimize cross-validation performance estimate. Grid search is a common search technique. In grid search, the space of candidate values of multiple hyperparameters is a cross-product of sets of candidate values of individual hyperparameters.

Grid search using cross-validation can be time-consuming. If an algorithm has five hyperparameters each with five candidate values, we use K=5 folds. We then complete a grid search by training $5^6 = 15625$ models. Fortunately, grid-search using cross-validation is an embarrassingly parallel procedure and all these models can be trained in parallel.

Prerequisites

- An [Azure account](#) (free trials are available).
- An installed copy of [Azure Machine Learning Workbench](#) following the [Install and create Quickstart](#) to install the Workbench and create accounts.
- This scenario assumes that you are running Azure ML Workbench on Windows 10 or MacOS with Docker engine locally installed.

- To run the scenario with a remote Docker container, provision Ubuntu Data Science Virtual Machine (DSVM) by following the [instructions](#). We recommend using a virtual machine with at least 8 cores and 28 Gb of memory. D4 instances of virtual machines have such capacity.
- To run this scenario with a Spark cluster, provision Spark HDInsight cluster by following these [instructions](#). We recommend having a cluster with the following configuration in both header and worker nodes:
 - four worker nodes
 - eight cores
 - 28 Gb of memory
 D4 instances of virtual machines have such capacity.

Troubleshooting: Your Azure subscription might have a quota on the number of cores that can be used. The Azure portal does not allow the creation of cluster with the total number of cores exceeding the quota. To find your quota, go in the Azure portal to the Subscriptions section, click on the subscription used to deploy a cluster and then click on **Usage+quotas**. Usually quotas are defined per Azure region and you can choose to deploy the Spark cluster in a region where you have enough free cores.

- Create an Azure storage account that is used for storing the dataset. Follow the [instructions](#) to create a storage account.

Data description

We use [TalkingData dataset](#). This dataset has events from the apps in cell phones. The goal is to predict gender and age category of cell phone user given the type of the phone and the events that the user generated recently.

Scenario structure

This scenario has multiple folders in GitHub repository. Code and configuration files are in **Code** folder, all documentation is in **Docs** folder and all images are **Images** folder. The root folder has README file that contains a brief summary of this scenario.

Getting started

Click on the Azure Machine Learning Workbench icon to run Azure Machine Learning Workbench and create a project from the "Distributed Tuning of Hyperparameters" template. You can find detailed instructions on how to create a new project in [Install and create Quickstart](#).

Configuration of execution environments

We show how to run our code in a remote Docker container and in a Spark cluster. We start with the description of the settings that are common to both environments.

We use [scikit-learn](#), [xgboost](#), and [azure-storage](#) packages that are not provided in the default Docker container of Azure Machine Learning Workbench. `azure-storage` package requires installation of [cryptography](#) and [azure](#) packages. To install these packages in the Docker container and in the nodes of Spark cluster, we modify `conda_dependencies.yml` file:

```

name: project_environment
channels:
- conda-forge
dependencies:
- python=3.5.2
- scikit-learn
- xgboost
- pip:
- cryptography
- azure
- azure-storage

```

The modified `conda_dependencies.yml` file is stored in `aml_config` directory of tutorial.

In the next steps, we connect execution environment to Azure account. Click File Menu from the top left corner of AML Workbench. And choose "Open Command Prompt". Then run in CLI

```
az login
```

You get a message

To sign in, use a web browser to open the page <https://aka.ms/devicelogin> and enter the code <code> to authenticate.

Go to this web page, enter the code and sign into your Azure account. After this step, run in CLI

```
az account list -o table
```

and find the Azure subscription ID that has your AML Workbench Workspace account. Finally, run in CLI

```
az account set -s <subscription ID>
```

to complete the connection to your Azure subscription.

In the next two sections, we show how to complete configuration of remote docker and Spark cluster.

Configuration of remote Docker container

To set up a remote Docker container, run in CLI

```
az ml computetarget attach remotedocker --name dsvm --address <IP address> --username <username> --password <password>
```

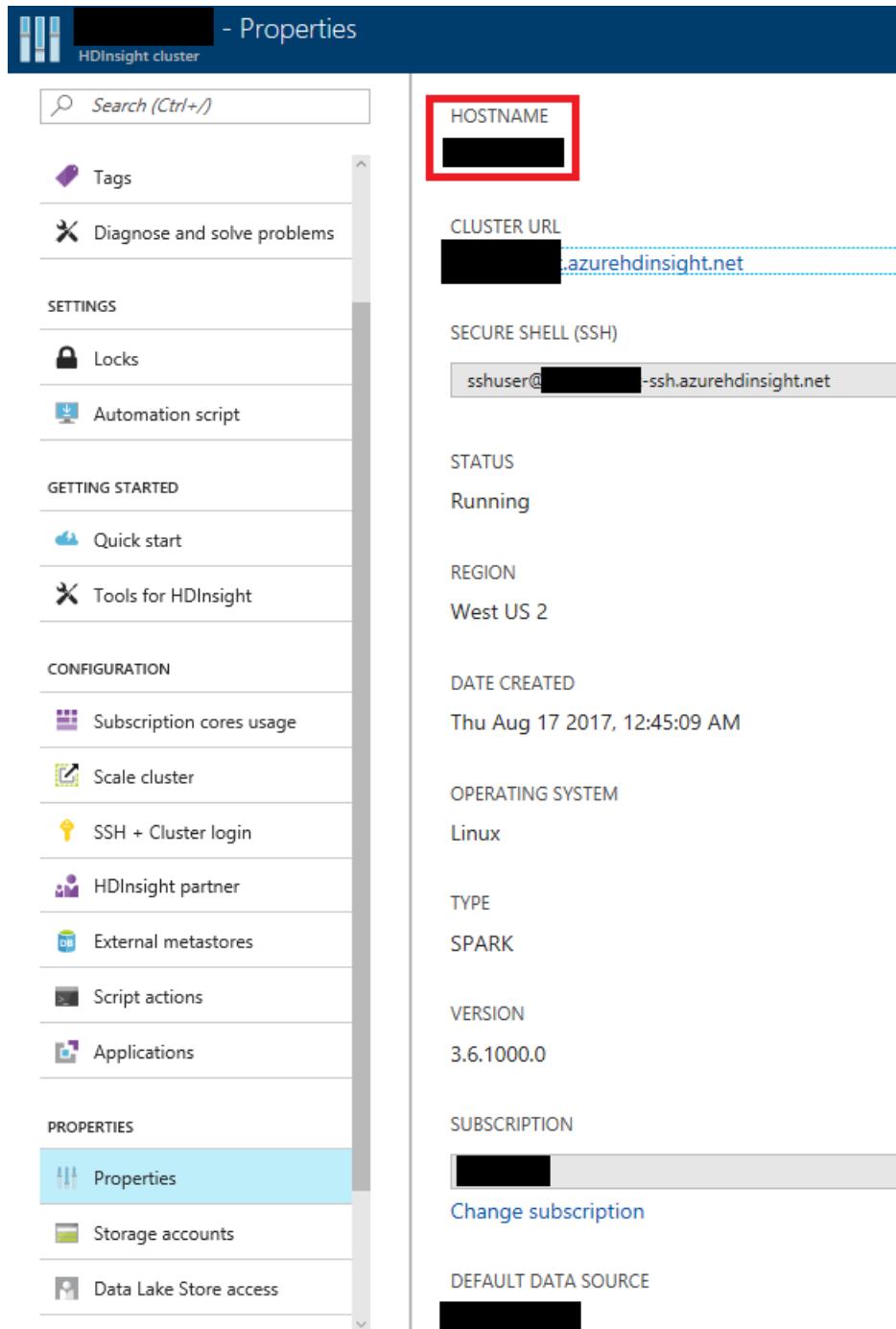
with IP address, user name and password in DSVM. IP address of DSVM can be found in Overview section of your DSVM page in Azure portal:

Configuration of Spark cluster

To set up Spark environment, run in CLI

```
az ml computetarget attach cluster --name spark --address <cluster name>-ssh.azurehdinsight.net --username <username> --password <password>
```

with the name of the cluster, cluster's SSH user name and password. The default value of SSH user name is `sshuser`, unless you changed it during provisioning of the cluster. The name of the cluster can be found in Properties section of your cluster page in Azure portal:



The screenshot shows the 'Properties' page for an 'HDInsight cluster'. The left sidebar lists various options like Tags, Diagnose and solve problems, Settings, Getting Started, Configuration, and Properties. The 'Properties' option is selected and highlighted with a blue background. The main content area displays cluster details under sections such as HOSTNAME, CLUSTER URL, SECURE SHELL (SSH), STATUS, REGION, DATE CREATED, OPERATING SYSTEM, TYPE, VERSION, SUBSCRIPTION, and DEFAULT DATA SOURCE. The 'HOSTNAME' field is explicitly highlighted with a red rectangular border.

Section	Value
HOSTNAME	[REDACTED]
CLUSTER URL	[REDACTED].azurehdinsight.net
SECURE SHELL (SSH)	sshuser@[REDACTED]-ssh.azurehdinsight.net
STATUS	Running
REGION	West US 2
DATE CREATED	Thu Aug 17 2017, 12:45:09 AM
OPERATING SYSTEM	Linux
TYPE	SPARK
VERSION	3.6.1000.0
SUBSCRIPTION	[REDACTED]
DEFAULT DATA SOURCE	[REDACTED]

We use spark-sklearn package to have Spark as an execution environment for distributed tuning of hyperparameters. We modified `spark_dependencies.yml` file to install this package when Spark execution environment is used:

```

configuration:
  #"spark.driver.cores": "8"
  #"spark.driver.memory": "5200m"
  #"spark.executor.instances": "128"
  #"spark.executor.memory": "5200m"
  #"spark.executor.cores": "2"

repositories:
  - "https://mmlspark.azureedge.net/maven"
  - "https://spark-packages.org/packages"

packages:
  - group: "com.microsoft.ml.spark"
    artifact: "mmlspark_2.11"
    version: "0.7.91"
  - group: "databricks"
    artifact: "spark-sklearn"
    version: "0.2.0"

```

The modified spark_dependencies.yml file is stored in aml_config directory of tutorial.

Data ingestion

The code in this scenario assumes that the data is stored in Azure blob storage. We show initially how to download data from Kaggle site to your computer and upload it to the blob storage. Then we show how to read the data from blob storage.

To download data from Kaggle, go to [dataset page](#) and click Download button. You will be asked to log in to Kaggle. After logging in, you will be redirected back to dataset page. Then download the first seven files in the left column by selecting them and clicking Download button. The total size of the downloaded files is 289 Mb. To upload these files to blob storage, create blob storage container 'dataset' in your storage account. You can do that by going to Azure page of your storage account, clicking Blobs and then clicking +Container. Enter 'dataset' as Name and click OK. The following screenshots illustrate these steps:

The screenshot shows the Azure Storage Account overview page. At the top, there's a search bar and navigation links for 'Open in Explorer', 'Move', and 'Delete storage account'. Below that, there's information about the resource group, status (Primary: Available), location (West US 2), and subscription details. A 'Subscription ID' field is also present. In the 'Services' section, the 'Blobs' service is highlighted with a red box. The 'Blobs' service is described as 'Object storage for understanding data' and includes options to 'View metrics', 'Configure CORS rules', and 'Setup custom domain'.

The screenshot shows the Blob service dashboard for a storage account. It includes fields for Container, Refresh, Storage account, Status (Primary: Available), Location (West US 2), Subscription (change), and Subscription ID. A search bar at the bottom is labeled "Search containers by prefix".

After that, select dataset container from the list and click Upload button. Azure portal allows you to upload multiple files concurrently. In "Upload blob" section click folder button, select all files from the dataset, click Open, and then click Upload. The following screenshot illustrates these steps:

The screenshot shows the Blob service dashboard with the "dataset" container selected. An "Upload blob" dialog is open, showing a list of files to be uploaded. The "Select a file" input field has a red box around it, indicating where to click. The "Upload" button is also highlighted.

Upload of the files takes several minutes, depending on your Internet connection.

In our code, we use [Azure Storage SDK](#) to download the dataset from blob storage to the current execution environment. The download is performed in `load_data()` function from `load_data.py` file. To use this code, you need to replace `<ACCOUNT_NAME>` and `<ACCOUNT_KEY>` with the name and primary key of your storage account that hosts the dataset. You can see the account name in the top left corner of your storage account's Azure page. To get account key, select Access Keys in Azure page of storage account (see the first screenshot in Data Ingestion section) and then copy the long string in the first row of key column:

The screenshot shows the "Access keys" section of the storage account settings. It displays two access keys: "key1" and "key2", each with a red box highlighting the key value. The "KEY" column for both keys is also highlighted with a red box.

The following code from `load_data()` function downloads a single file:

```

from azure.storage.blob import BlockBlobService

# Define storage parameters
ACCOUNT_NAME = "<ACCOUNT_NAME>"
ACCOUNT_KEY = "<ACCOUNT_KEY>"
CONTAINER_NAME = "dataset"

# Define blob service
my_service = BlockBlobService(account_name=ACCOUNT_NAME, account_key=ACCOUNT_KEY)

# Load blob
my_service.get_blob_to_path(CONTAINER_NAME, 'app_events.csv.zip', 'app_events.csv.zip')

```

Notice that you do not need to run `load_data.py` file manually. It is called from other files later on.

Feature engineering

The code for computing all features is in `feature_engineering.py` file. You do not need to run `feature_engineering.py` file manually. Later on it will be called from other files.

We create multiple feature sets:

- One-hot encoding of brand and model of the cell phone (`one_hot_brand_model` function)
- Fraction of events generated by user in each weekday (`weekday_hour_features` function)
- Fraction of events generated by user in each hour (`weekday_hour_features` function)
- Fraction of events generated by user in each combination of weekday and hour (`weekday_hour_features` function)
- Fraction of events generated by user in each app (`one_hot_app_labels` function)
- Fraction of events generated by user in each app label (`one_hot_app_labels` function)
- Fraction of events generated by user in each app category (`text_category_features` function)
- Indicator features for categories of apps that were used to generated events (`one_hot_category` function)

These features were inspired by Kaggle kernel [A linear model on apps and labels](#).

The computation of these features requires significant amount of memory. Initially we tried to compute features in the local environment with 16-GB RAM. We were able to compute the first four sets of features, but received 'Out of memory' error when computing the fifth feature set. The computation of the first four feature sets is in `singleVMsmall.py` file and it can be executed in the local environment by running

```
az ml experiment submit -c local .\singleVMsmall.py
```

in CLI window.

Since local environment is too small for computing all feature sets, we switch to remote DSVM that has larger memory. The execution inside DSVM is done inside Docker container that is managed by AML Workbench. Using this DSVM we are able to compute all features and train models and tune hyperparameters (see the next section). `singleVM.py` file has complete feature computation and modeling code. In the next section, we will show how to run `singleVM.py` in remote DSVM.

Tuning hyperparameters using remote DSVM

We use [xgboost](#) implementation [1] of gradient tree boosting. We also use [scikit-learn](#) package to tune hyperparameters of xgboost. Although xgboost is not part of scikit-learn package, it implements scikit-learn API and hence can be used together with hyperparameter tuning functions of scikit-learn.

Xgboost has eight hyperparameters, described [here](#):

- `n_estimators`

- max_depth
- reg_alpha
- reg_lambda
- colsample_by_tree
- learning_rate
- colsample_by_level
- subsample
- objective

Initially, we use remote DSVM and tune hyperparameters from a small grid of candidate values:

```
tuned_parameters = [{n_estimators': [300,400], 'max_depth': [3,4], 'objective': ['multi:softprob'],
'reg_alpha': [1], 'reg_lambda': [1], 'colsample_bytree': [1], 'learning_rate': [0.1], 'colsample_bylevel': [0.1], 'subsample': [0.5]}]
```

This grid has four combinations of values of hyperparameters. We use 5-fold cross validation, resulting in $4 \times 5 = 20$ runs of xgboost. To measure performance of the models, we use negative log loss metric. The following code finds the values of hyperparameters from the grid that maximize the cross-validated negative log loss. The code also uses these values to train the final model over the full training set:

```
clf = XGBClassifier(seed=0)
metric = 'neg_log_loss'

clf_cv = GridSearchCV(clf, tuned_parameters, scoring=metric, cv=5, n_jobs=8)
model = clf_cv.fit(X_train,y_train)
```

After creating the model, we save the results of the hyperparameter tuning. We use logging API of AML Workbench to save the best values of hyperparameters and corresponding cross-validated estimate of the negative log loss:

```
from azureml.logging import get_azureml_logger

# initialize logger
run_logger = get_azureml_logger()

...
run_logger.log(metric, float(clf_cv.best_score_))

for key in clf_cv.best_params_.keys():
    run_logger.log(key, clf_cv.best_params_[key])
```

We also create sweeping_results.txt file with cross-validated, negative log losses of all combinations of hyperparameter values in the grid.

```
if not path.exists('./outputs'):
    makedirs('./outputs')
outfile = open('./outputs/sweeping_results.txt','w')

print("metric = ", metric, file=outfile)
for i in range(len(model.grid_scores_)):
    print(model.grid_scores_[i], file=outfile)
outfile.close()
```

This file is stored in a special ./outputs directory. Later on we show how to download it.

Before running singleVM.py in remote DSVM for the first time, we create a Docker container there by running

```
az ml experiment prepare -c dsvm
```

in CLI windows. Creation of Docker container takes several minutes. After that we run singleVM.py in DSVM:

```
az ml experiment submit -c dsvm .\singleVM.py
```

This command finishes in 1 hour 38 minutes when DSVM has 8 cores and 28 Gb of memory. The logged values can be viewed in Run History window of AML Workbench:



By default Run History window shows values and graphs of the first 1-2 logged values. To see the full list of the chosen values of hyperparameters, click on the settings icon marked with red circle in the previous screenshot. Then, select the hyperparameters to be shown in the table. Also, to select the graphs that are shown in the top part of Run History window, click on the setting icon marked with blue circle and select the graphs from the list.

The chosen values of hyperparameters can also be examined in Run Properties window:

The figure shows the AML Workbench Run Properties window. On the left is a table of hyperparameters and their values. On the right is an 'Output Files' section with a 'Download' button and a list of files: sweeping_results.txt (selected) and sdk_debug.txt.

Status	Completed
Start Time	Sep 19, 2017, 10:21:43 AM
Duration	1:38:44
Target	dsvm
Run Id	distributed_tuning_1505830893746
Run Number	1
Script Name	singleVM.py
Git Commit Hash	a44b97e8b7b5214869a22afe3e32d945d9d86061
neg_log_loss	-2.285299172456161
colsample_bytree	1
learning_rate	0.1
subsample	0.5
n_estimators	400
reg_alpha	1
colsample_bylevel	0.1
objective	multisoftprob
reg_lambda	1
max_depth	4

Output Files

Download Promote

sweeping_results.txt
 sdk_debug.txt

In the top right corner of Run Properties window, there is a section Output Files with the list of all files that were created in '\output' folder. sweeping_results.txt can be downloaded from there by selecting it and clicking Download button. sweeping_results.txt should have the following output:

```

metric = neg_log_loss
mean: -2.29096, std: 0.03748, params: {'colsample_bytree': 1, 'learning_rate': 0.1, 'subsample': 0.5,
'n_estimators': 300, 'reg_alpha': 1, 'objective': 'multi:softprob', 'colsample_bylevel': 0.1, 'reg_lambda': 1,
'max_depth': 3}
mean: -2.28712, std: 0.03822, params: {'colsample_bytree': 1, 'learning_rate': 0.1, 'subsample': 0.5,
'n_estimators': 400, 'reg_alpha': 1, 'objective': 'multi:softprob', 'colsample_bylevel': 0.1, 'reg_lambda': 1,
'max_depth': 3}
mean: -2.28706, std: 0.03863, params: {'colsample_bytree': 1, 'learning_rate': 0.1, 'subsample': 0.5,
'n_estimators': 300, 'reg_alpha': 1, 'objective': 'multi:softprob', 'colsample_bylevel': 0.1, 'reg_lambda': 1,
'max_depth': 4}
mean: -2.28530, std: 0.03927, params: {'colsample_bytree': 1, 'learning_rate': 0.1, 'subsample': 0.5,
'n_estimators': 400, 'reg_alpha': 1, 'objective': 'multi:softprob', 'colsample_bylevel': 0.1, 'reg_lambda': 1,
'max_depth': 4}

```

Tuning hyperparameters using Spark cluster

We use Spark cluster to scale out tuning hyperparameters and use larger grid. Our new grid is

```

tuned_parameters = [{n_estimators: [300,400], max_depth: [3,4], objective: ['multi:softprob'],
'reg_alpha': [1], 'reg_lambda': [1], 'colsample_bytree': [1], 'learning_rate': [0.1], 'colsample_bylevel':
[0.01, 0.1], 'subsample': [0.5, 0.7]}]

```

This grid has 16 combinations of values of hyperparameters. Since we use 5-fold cross validation, we run xgboost 16x5=80 times.

scikit-learn package does not have a native support of tuning hyperparameters using Spark cluster. Fortunately, [spark-sklearn](#) package from Databricks fills this gap. This package provides GridSearchCV function that has almost the same API as GridSearchCV function in scikit-learn. To use spark-sklearn and tune hyperparameters using Spark we need to create a Spark context

```

from pyspark import SparkContext
sc = SparkContext.getOrCreate()

```

Then we replace

```

from sklearn.model_selection import GridSearchCV

```

with

```

from spark_sklearn import GridSearchCV

```

Also we replace the call to GridSearchCV from scikit-learn to the one from spark-sklearn:

```

clf_cv = GridSearchCV(sc = sc, param_grid = tuned_parameters, estimator = clf, scoring=metric, cv=5)

```

The final code for tuning hyperparameters using Spark is in distributed_sweep.py file. The difference between singleVM.py and distributed_sweep.py is in definition of grid and additional four lines of code. Notice also that due to AML Workbench services, the logging code does not change when changing execution environment from remote DSVM to Spark cluster.

Before running distributed_sweep.py in Spark cluster for the first time, we need to install Python packages there. This can be achieved by running

```
az ml experiment prepare -c spark
```

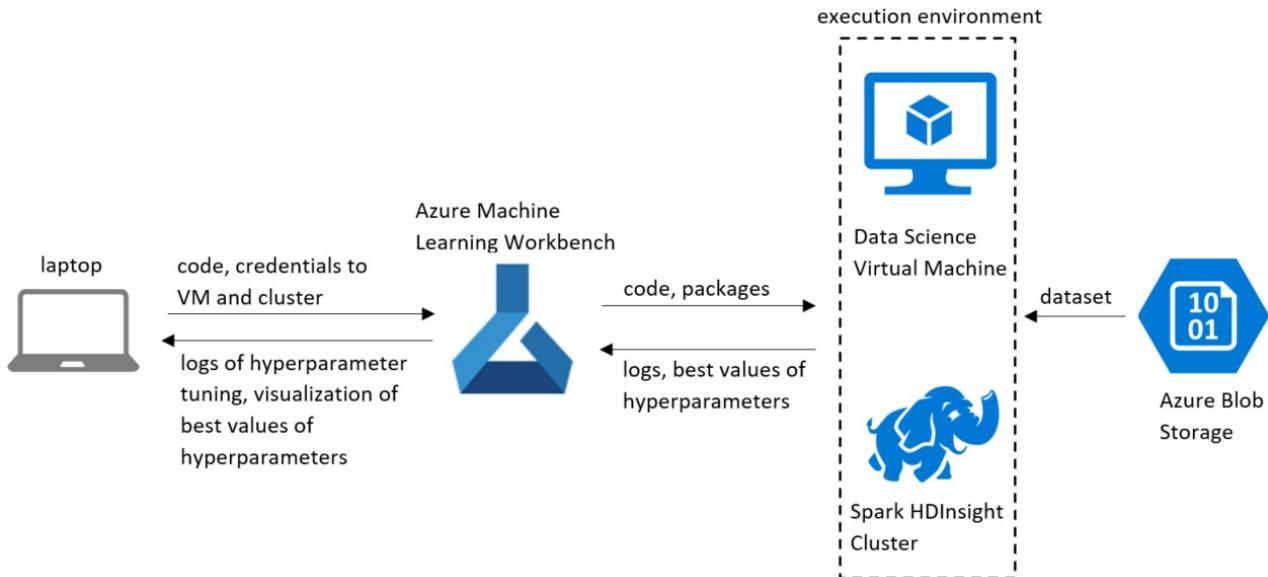
in CLI windows. This installation takes several minutes. After that we run `distributed_sweep.py` in Spark cluster:

```
az ml experiment submit -c spark .\distributed_sweep.py
```

This command finishes in 1 hour 6 minutes when Spark cluster has 6 worker nodes with 28 Gb of memory. The results of hyperparameter-tuning can be accessed in Azure Machine Learning Workbench the same way as remote DSVM execution. (namely logs, best values of hyperparameters, and `sweeping_results.txt` file)

Architecture diagram

The following diagram shows the overall workflow:



Conclusion

In this scenario, we showed how to use Azure Machine Learning Workbench to perform tuning of hyperparameters in remote virtual machines and Spark clusters. We saw that Azure Machine Learning Workbench provides tools for easy configuration of execution environments. It also allows easily switching between them.

References

- [1] T. Chen and C. Guestrin. [XGBoost: A Scalable Tree Boosting System](#). KDD 2016.

Customer churn prediction using Azure Machine Learning

9/24/2018 • 3 minutes to read • [Edit Online](#)

NOTE

This article is deprecated. Support for earlier versions of this service will end incrementally. [View the support timeline](#). Start using the latest version with this [quickstart](#).

On average, keeping existing customers is five times cheaper than the cost of recruiting new ones. As a result, marketing executives often find themselves trying to estimate the likelihood of customer churn and finding the necessary actions to minimize the churn rate.

The aim of this solution is to demonstrate predictive churn analytics using Azure Machine Learning Workbench. This solution provides an easy to use template to develop churn predictive data pipelines for retailers. The template can be used with different datasets and different definitions of churn. The aim of the hands-on example is to:

1. Understand Azure Machine Learning Workbench's Data Preparation tools to clean and ingest customer relationship data for churn analytics.
2. Perform feature transformation to handle noisy heterogeneous data.
3. Integrate third-party libraries (such as `scikit-learn` and `azureml`) to develop Bayesian and Tree-based classifiers for predicting churn.
4. Deploy.

Link of the gallery GitHub repository

Following is the link to the public GitHub repository where all the code is hosted:

<https://github.com/Azure/MachineLearningSamples-ChurnPrediction>

Use case overview

Companies need an effective strategy for managing customer churn. Customer churn includes customers stopping the use of a service, switching to a competitor service, switching to a lower-tier experience in the service or reducing engagement with the service.

In this use case, we look at data from French telecom company Orange to identify customers who are likely to churn in the near term in order to improve the service and create custom outreach campaigns that help retain customers.

Telecom companies face a competitive market. Many carriers lose revenue from postpaid customers due to churn. Hence the ability to accurately identify customer churn can be a huge competitive advantage.

Some of the factors contributing to telecom customer churn include:

- Perceived frequent service disruptions
- Poor customer service experiences in online/retail stores
- Offers from other competing carriers (better family plan, data plan, etc.).

In this solution, we will use a concrete example of building a predictive customer churn model for telecom companies.

Prerequisites

- An [Azure account](#) (free trials are available)
- An installed copy of [Azure Machine Learning Workbench](#) following the [quick start installation guide](#) to install the program and create a workspace
- For operationalization, it is best if you have Docker engine installed and running locally. If not, you can use the cluster option but be aware that running an Azure Container Service (ACS) can be expensive.
- This Solution assumes that you are running Azure Machine Learning Workbench on Windows 10 with Docker engine locally installed. If you are using macOS the instruction is largely the same.

Create a new Workbench project

Create a new project using this example as a template:

1. Open Azure Machine Learning Workbench
2. On the **Projects** page, click the + sign and select **New Project**
3. In the **Create New Project** pane, fill in the information for your new project
4. In the **Search Project Templates** search box, type "Customer Churn Prediction" and select the template
5. Click **Create**

Data description

The data set used in the solution is from the SIDKDD 2009 competition. It is called

`CATelcoCustomerChurnTrainingSample.csv` and is located in the `data` folder. The dataset consists of heterogeneous noisy data (numerical/categorical variables) from French Telecom company Orange and is anonymized.

The variables capture customer demographic information, call statistics (such as average call duration, call failure rate, etc.), contract information, complaint statistics. Churn variable is binary (0 - did not churn, 1 - did churn).

Scenario structure

The folder structure is arranged as follows:

data: Contains the dataset used in the solution

docs: Contains all the hands-on labs

The order of Hands-on Labs to carry out the solution is as follows:

1. Data Preparation: The main file related to Data Preparation in the data folder is
`CATelcoCustomerChurnTrainingSample.csv`
2. Modeling and Evaluation: The main file related to modeling and evaluation in the root folder is
`CATelcoCustomerChurnModeling.py`
3. Modeling and Evaluation without .dprep: The main file for this task in the root folder is
`CATelcoCustomerChurnModelingWithoutDprep.py`
4. Operationalization: The main files for deployment are the model (`model.pkl`) and `churn_schema_gen.py`

ORDER	FILE NAME	REALTED FILES
1	DataPreparation.md	'data/CATelcoCustomerChurnTrainingSa mple.csv'
2	ModelingAndEvaluation.md	'CATelcoCustomerChurnModeling.py'
3	ModelingAndEvaluationWithoutDprep.md	'CATelcoCustomerChurnModelingWitho utDprep.py'
4	Operationalization.md	'model.pkl' 'churn_schema_gen.py'

Follow the Labs in the sequential manner described above.

Conclusion

This hands on scenario demonstrated how to perform churn prediction using Azure Machine Learning Workbench. We first performed data cleaning to handle noisy and heterogeneous data, followed by feature engineering using Data Preparation tools. We then used open source machine learning tools to create and evaluate a classification model, then used local docker container to deploy the model making it ready for production.

Sentiment Analysis using Deep Learning with Azure Machine Learning

9/24/2018 • 2 minutes to read • [Edit Online](#)

NOTE

This article is deprecated. Support for earlier versions of this service will end incrementally. [View the support timeline](#). Start using the latest version with this [quickstart](#).

Sentiment analysis is a well-known task in the realm of natural language processing. Given a set of texts, the aim is to determine the sentiment of that text. The objective of this solution is to use Deep Learning for predicting sentiment from movie reviews.

The solution is located at <https://github.com/Azure/MachineLearningSamples-SentimentAnalysis>

Link to the Gallery GitHub repository

Follow this link to the public GitHub repository:

<https://github.com/Azure/MachineLearningSamples-SentimentAnalysis>

Use case overview

The explosion of data and the proliferation of mobile devices have created lots of opportunities for customers to express their feelings and attitudes about anything and everything at any time. This opinion or "sentiment" is often generated through social channels in the form of reviews, chats, shares, likes, tweets, etc. The sentiment can be invaluable for businesses looking to improve products and services, make more informed decisions, and better promote their brands.

To get value from sentiment analysis, businesses must have the ability to mine vast stores of unstructured social data for actionable insights. In this sample, we develop deep learning models for performing sentiment analysis of movie reviews using AMLWorkbench

Prerequisites

- An [Azure account](#) (free trials are available).
- An installed copy of [Azure Machine Learning Workbench](#) following the [quick start installation guide](#) to install the program and create a workspace.
- For operationalization, it is best if you have Docker engine installed and running locally. If not, you can use the cluster option. However, running an Azure Container Service (ACS) can be expensive.
- This Solution assumes that you are running Azure Machine Learning Workbench on Windows 10 with Docker engine locally installed. On a macOS, the instruction is largely the same.

Data description

The dataset used for this sample is a small hand-crafted dataset and is located in the [data folder](#).

The first column contains movie reviews and the second column contains their sentiment (0 - negative and 1 -

positive). The dataset is merely used for demonstration purposes but typically to get robust sentiment scores, you need a large dataset. For example, the [IMDB Movie reviews sentiment classification problem](#) from Keras consists of a dataset of 25,000 movies reviews from IMDB, labeled by sentiment (positive or negative). The intention of this lab is to show you how to perform sentiment analysis using deep learning with AMLWorkbench.

Scenario structure

The folder structure is arranged as follows:

1. All the code related to sentiment analysis using AMLWorkbench is in the root folder
2. data: Contains the dataset used in the solution
3. docs: Contains all the hands-on labs

The order of Hands-on Labs to carry out the solution is as follows:

ORDER	FILE NAME	RELATED FILES
1	SentimentAnalysisDataPreparation.md	'data/sampleReviews.txt'
2	SentimentAnalysisModelingKeras.md	'SentimentExtraction.py'
3	SentimentAnalysisOperationalization.md	'Operationalization'

Conclusion

In conclusion, this solution introduces you to using Deep Learning to perform sentiment analysis with the Azure Machine Learning Workbench. We also operationalize using HDF5 models.

Biomedical entity recognition using Team Data Science Process (TDSP) Template

11/20/2018 • 12 minutes to read • [Edit Online](#)

NOTE

This article is deprecated. Support for earlier versions of this service will end incrementally. [View the support timeline](#). Start using the latest version with this [quickstart](#).

Entity extraction is a subtask of information extraction (also known as [Named-entity recognition \(NER\)](#), entity chunking, and entity identification). The aim of this real-world scenario is to highlight how to use Azure Machine Learning Workbench to solve a complicated Natural Language Processing (NLP) task such as entity extraction from unstructured text:

1. How to train a neural word embeddings model on a text corpus of about 18 million PubMed abstracts using [Spark Word2Vec implementation](#).
2. How to build a deep Long Short-Term Memory (LSTM) recurrent neural network model for entity extraction on a GPU-enabled Azure Data Science Virtual Machine (GPU DS VM) on Azure.
3. Demonstrate that domain-specific word embeddings model can outperform generic word embeddings models in the entity recognition task.
4. Demonstrate how to train and operationalize deep learning models using Azure Machine Learning Workbench.
5. Demonstrate the following capabilities within Azure Machine Learning Workbench:
 - Instantiation of [Team Data Science Process \(TDSP\) structure and templates](#)
 - Automated management of your project dependencies including the download and the installation
 - Execution of Python scripts on different compute environments
 - Run history tracking for Python scripts
 - Execution of jobs on remote Spark compute contexts using HDInsight Spark 2.1 clusters
 - Execution of jobs in remote GPU VMs on Azure
 - Easy operationalization of deep learning models as web services on Azure Container Services (ACS)

Use case overview

Biomedical named entity recognition is a critical step for complex biomedical NLP tasks such as:

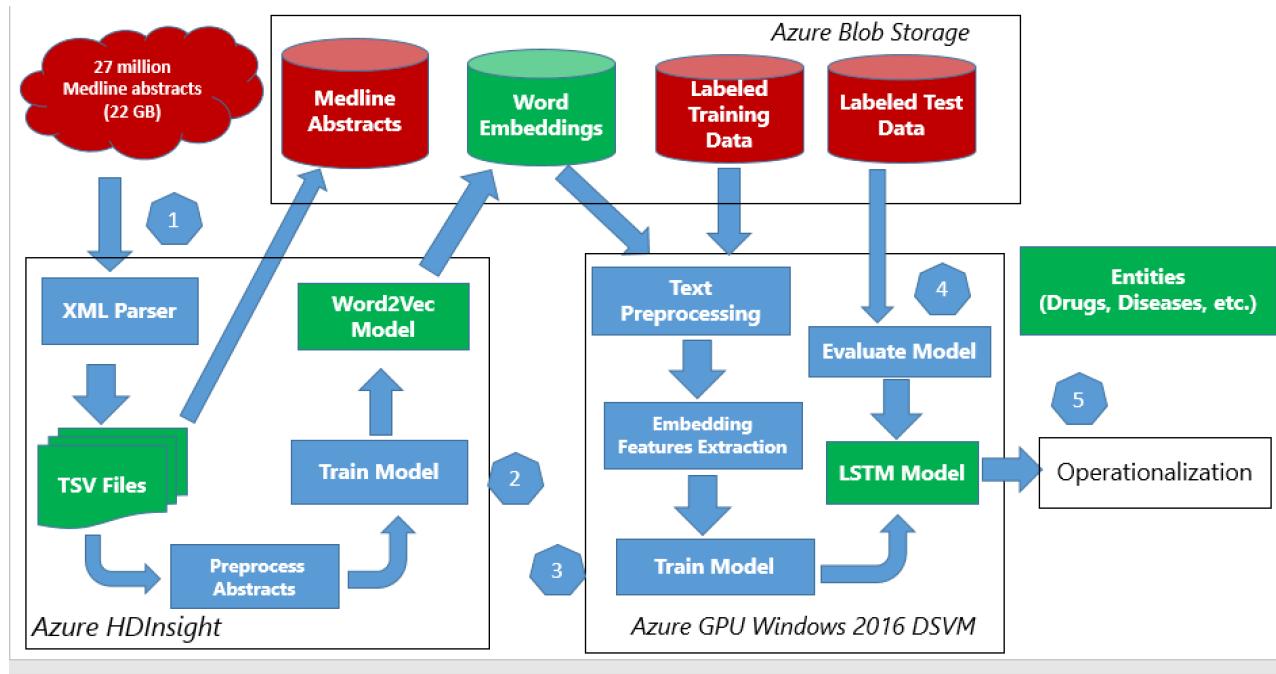
- Extracting the mentions of named entities such diseases, drugs, chemicals, and symptoms from electronic medical or health records.
- Drug discovery
- Understanding the interactions between different entity types such as drug-drug interaction, drug-disease relationship, and gene-protein relationship.

Our use case scenario focuses on how a large amount of unstructured data corpus such as Medline PubMed abstracts can be analyzed to train a word embedding model. Then the output embeddings are considered as automatically generated features to train a neural entity extractor.

Our results show that the biomedical entity extraction model training on the domain-specific word embedding

features outperforms the model trained on the generic feature type. The domain-specific model can detect 7012 entities correctly (out of 9475) with F1-score of 0.73 compared to 5274 entities with F1-score of 0.61 for the generic model.

The following figure shows the architecture that was used to process data and train models.



Data description

1. Word2Vec model training data

We first downloaded the raw MEDLINE abstract data from [MEDLINE](#). The data is publicly available in the form of XML files on their [FTP server](#). There are 892 XML files available on the server and each of the XML files has the information of 30,000 articles. More details about the data collection step are provided in the Project Structure section. The fields present in each file are

```

abstract
affiliation
authors
country
delete: boolean if False means paper got updated so you might have two XMLs for the same paper.
file_name
issn_linking
journal
keywords
medline_ta: this is abbreviation of the journal name
mesh_terms: list of MeSH terms
nlm_unique_id
other_id: Other IDs
pmc: Pubmed Central ID
pmid: Pubmed ID
pubdate: Publication date
title
  
```

2. LSTM model training data

The neural entity extraction model has been trained and evaluated on publicly available datasets. To obtain a detailed description about these datasets, you could refer to the following sources:

- [Bio-Entity Recognition Task at BioNLP/NLPBA 2004](#)
- [BioCreative V CDR task corpus](#)

- Semeval 2013 - Task 9.1 (Drug Recognition)

Link to the Azure gallery GitHub repository

Following is the link to the public GitHub repository of the real-world scenario that contains the code and more detailed description:

<https://github.com/Azure/MachineLearningSamples-BiomedicalEntityExtraction>

Prerequisites

- An Azure [subscription](#)
- Azure Machine Learning Workbench. See [installation guide](#). Currently the Azure Machine Learning Workbench can be installed on the following operating systems only:
 - Windows 10 or Windows Server 2016
 - macOS Sierra (or newer)

Azure services

- [HDInsight Spark cluster](#) version Spark 2.1 on Linux (HDI 3.6) for scale-out computation. To process the full amount of MEDLINE abstracts discussed below, you need the minimum configuration of:
 - Head node: [D13_V2](#) size
 - Worker nodes: At least 4 of [D12_V2](#). In our work, we used 11 worker nodes of D12_V2 size.
- [NC6 Data Science Virtual Machine \(DSVM\)](#) for scale-up computation.

Python packages

All the required dependencies are defined in the aml_config/conda_dependencies.yml file under the scenario project folder. The dependencies defined in this file are automatically provisioned for runs against docker, VM, and HDI cluster targets. For details about the Conda environment file format, refer to [here](#).

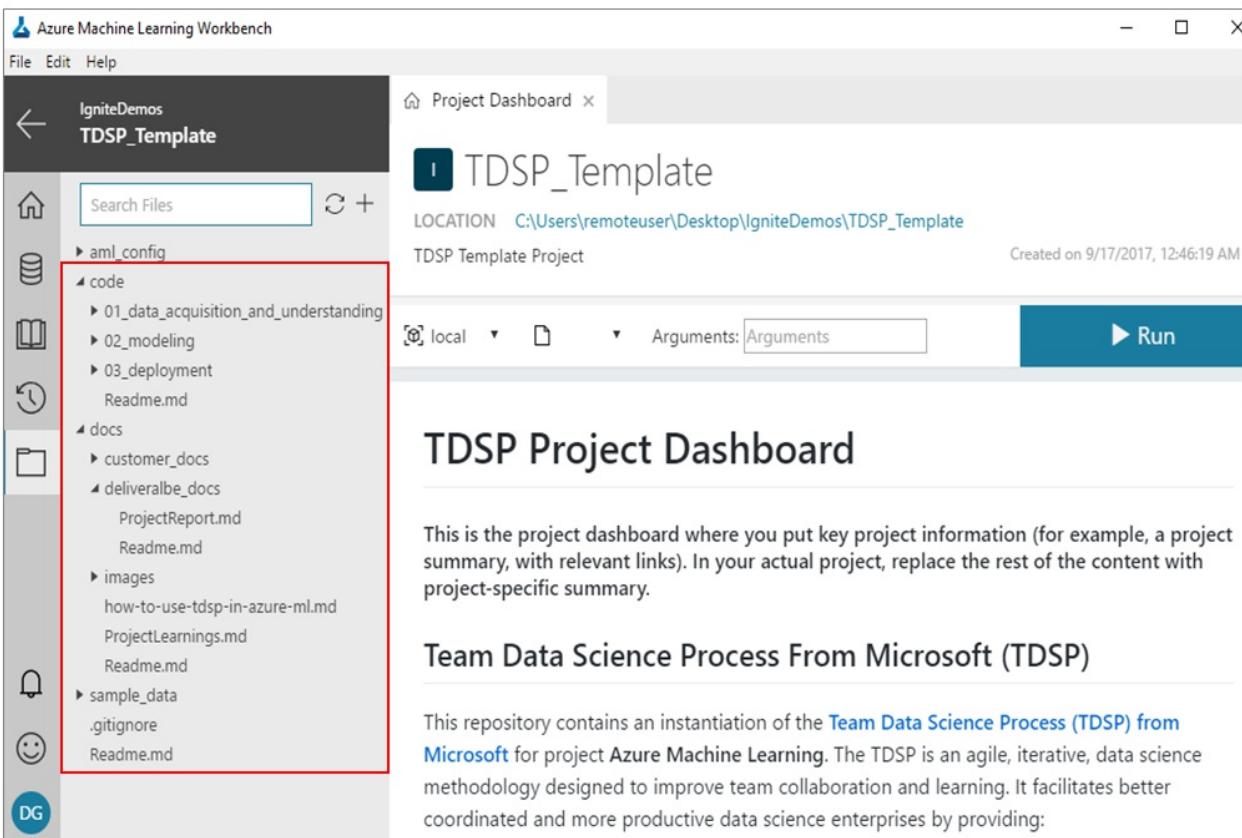
- [TensorFlow](#)
- [CNTK 2.0](#)
- [Keras](#)
- [NLTK](#)
- [Fastparquet](#)

Basic instructions for Azure Machine Learning (AML) workbench

- [Overview](#)
- [Installation](#)
- [Using TDSP](#)
- [How to read and write files](#)
- [How to use Jupyter Notebooks](#)
- [How to use GPU](#)

Scenario structure

For the scenario, we use the TDSP project structure and documentation templates (Figure 1), which follows the [TDSP lifecycle](#). Project is created based on instructions provided [here](#).



The step-by-step data science workflow is as follows:

1. Data acquisition and understanding

See [Data Acquisition and Understanding](#).

The raw MEDLINE corpus has a total of 27 million abstracts where about 10 million articles have an empty abstract field. Azure HDInsight Spark is used to process big data that cannot be loaded into the memory of a single machine as a [Pandas DataFrame](#). First, the data is downloaded into the Spark cluster. Then the following steps are executed on the [Spark DataFrame](#):

- parse the XML files using Medline XML Parser
- preprocess the abstract text including sentence splitting, tokenization, and case normalization.
- exclude articles where abstract field is empty or has short text
- create the word vocabulary from the training abstracts
- train the word embedding neural model. For more information, see [GitHub code link](#) to get started.

After parsing XML files, data has the following format:

abstract	Association of dilution hyponatremia and normal natriuresis produce an inappropriate secretion of antidiuretic hormone. In this pulmonary tuberculosis case we report manifestations along with the other biological criteria of inappropriate secretion of antidiuretic hormone. All disturbances cease when pulmonary tuberculosis is cured. This syndrome may be due either to disturbance in the regulatory apparatus of antidiuretic hormone secretion or to the antidiuretic principle in tubercular lung tissue.
affiliation	
author	P Peltier; R Grossette; G Dabouis; J Coroller
country	FRANCE
delete	FALSE
file_name	medline16n0189.xml.gz
issn_linking	
journal	La semaine des hôpitaux : organe fondé par l'Association d'enseignement médical des hôpitaux de Paris
keywords	
medline_ta	Sem Hop
mesh_terms	Aged; Humans; Hyponatremia; Inappropriate ADH Syndrome; Male; Natriuresis; Tuberculosis, Pulmonary
nlm_unique_id	9410059
other_id	
pmc	
pmid	6244674
pubdate	1980
title	[Dilution hyponatremia and conserved natriuresis in pulmonary tuberculosis (author's transl)].

The neural entity extraction model has been trained and evaluated on publicly available datasets. To obtain a detailed description about these datasets, you could refer to the following sources:

- [Bio-Entity Recognition Task at BioNLP/NLPBA 2004](#)
- [BioCreative V CDR task corpus](#)
- [Semeval 2013 - Task 9.1 \(Drug Recognition\)](#)

2. Modeling

See [Modeling](#).

Modeling is the stage where we show how you can use the data downloaded in the previous section for training your own word embedding model and use it for other downstream tasks. Although we are using the PubMed data, the pipeline to generate the embeddings is generic and can be reused to train word embeddings for any other domain. For embeddings to be an accurate representation of the data, it is essential that the word2vec is trained on a large amount of data. Once we have the word embeddings ready, we can train a deep neural network model that uses the learned embeddings to initialize the Embedding layer. We mark the embedding layer as non-trainable but that is not mandatory. The training of the word embedding model is unsupervised and hence we are able to take advantage of unlabeled texts. However, the training of the entity recognition model is a supervised learning task and its accuracy depends on the amount and the quality of a manually-annotated data.

2.1. Feature generation

See [Feature generation](#).

Word2Vec is the word embedding unsupervised learning algorithm that trains a neural network model from an unlabeled training corpus. It produces a continuous vector for each word in the corpus that represents its semantic information. These models are simple neural networks with a single hidden layer. The word vectors/embeddings are learned by backpropagation and stochastic gradient descent. There are two types of word2vec models, namely, the Skip-Gram and the continuous-bag-of-words (check [here](#) for more details). Since we are using the MLLib's implementation of the word2vec, which supports the Skip-gram model, we briefly describe it here (image taken from [link](#)):

Skip-gram Model

Input: A single word W_i

Output: Words in W_i 's context $\{W_{o1}, \dots, W_{oC}\}$ where C is the window size

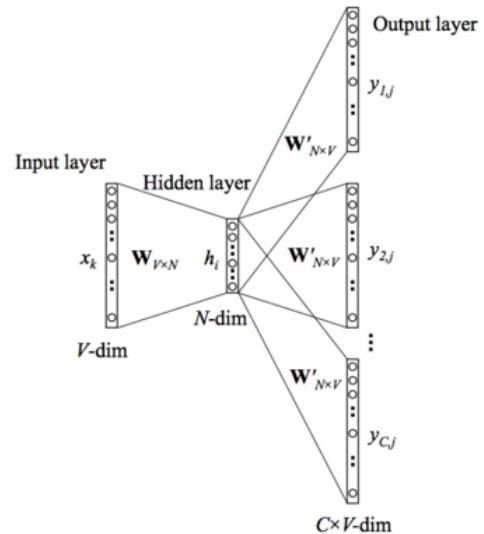
Example: I love deep **learning** for NLP

Input: learning

Output: "love", "deep", "for", "NLP" for window size of 2

$$\text{Objective Function: } p(w_{c,j} = w_{O,c} | w_I) = y_{c,j} = \frac{\exp(u_{c,j})}{\sum_{j'=1}^V \exp(u_{j'})}$$

$$\begin{aligned} \text{Gradient Descent Updates: } w_{ij}^{(new)} &= w_{ij}^{(old)} - \eta \cdot \sum_{c=1}^C (y_{c,j} - t_{c,j}) \cdot h_i \\ w_{ij}^{(new)} &= w_{ij}^{(old)} - \eta \cdot \sum_{j=1}^V \sum_{c=1}^C (y_{c,j} - t_{c,j}) \cdot w'_{ij} \cdot x_j \end{aligned}$$



Optimization using Hierarchical SoftMax and Negative Sampling

The model uses Hierarchical Softmax and Negative sampling to optimize the performance. Hierarchical SoftMax (H-SoftMax) is an approximation inspired by binary trees. H-SoftMax essentially replaces the flat SoftMax layer with a hierarchical layer that has the words as leaves. This allows us to decompose calculating the probability of one word into a sequence of probability calculations, which saves us from having to calculate the expensive normalization over all words. Since a balanced binary tree has a depth of $\log_2(|V|)$ (V is the Vocabulary), we only need to evaluate at most $\log_2(|V|)$ nodes to obtain the final probability of a word. The probability of a word w given its context c is then simply the product of the probabilities of taking right and left turns respectively that lead to its leaf node. We can build a Huffman Tree based on the frequency of the words in the dataset to ensure that more frequent words get shorter representations. For more information, see [this link](#). Image taken from [here](#).

Visualization

Once we have the word embeddings, we would like to visualize them and see the relationship between semantically similar words.

```
model.findSynonyms("cancer", 20).select("word").head(20) #Returns types of Cancers, hormones responsible for Cancer etc.

[Row(word=u'crc'), Row(word=u'colorectal'), Row(word=u'eoc'), Row(word=u'breast'), Row(word=u'hormone-refractory'), Row(word=u'nsclc'), Row(word=u'crpc'), Row(word=u'prostate'), Row(word=u'cancers'), Row(word=u'early-stage'), Row(word=u'carcinoma'), Row(word=u'adenocarcinoma'), Row(word=u'her2-positive'), Row(word=u'head-and-neck'), Row(word=u'hcc'), Row(word=u'metastatic'), Row(word=u'hnscc'), Row(word=u'gbc'), Row(word=u'mcrc'), Row(word=u'tnbc')]

model.findSynonyms("brain", 20).select("word").head(20) # Returns Different Parts of the Brain

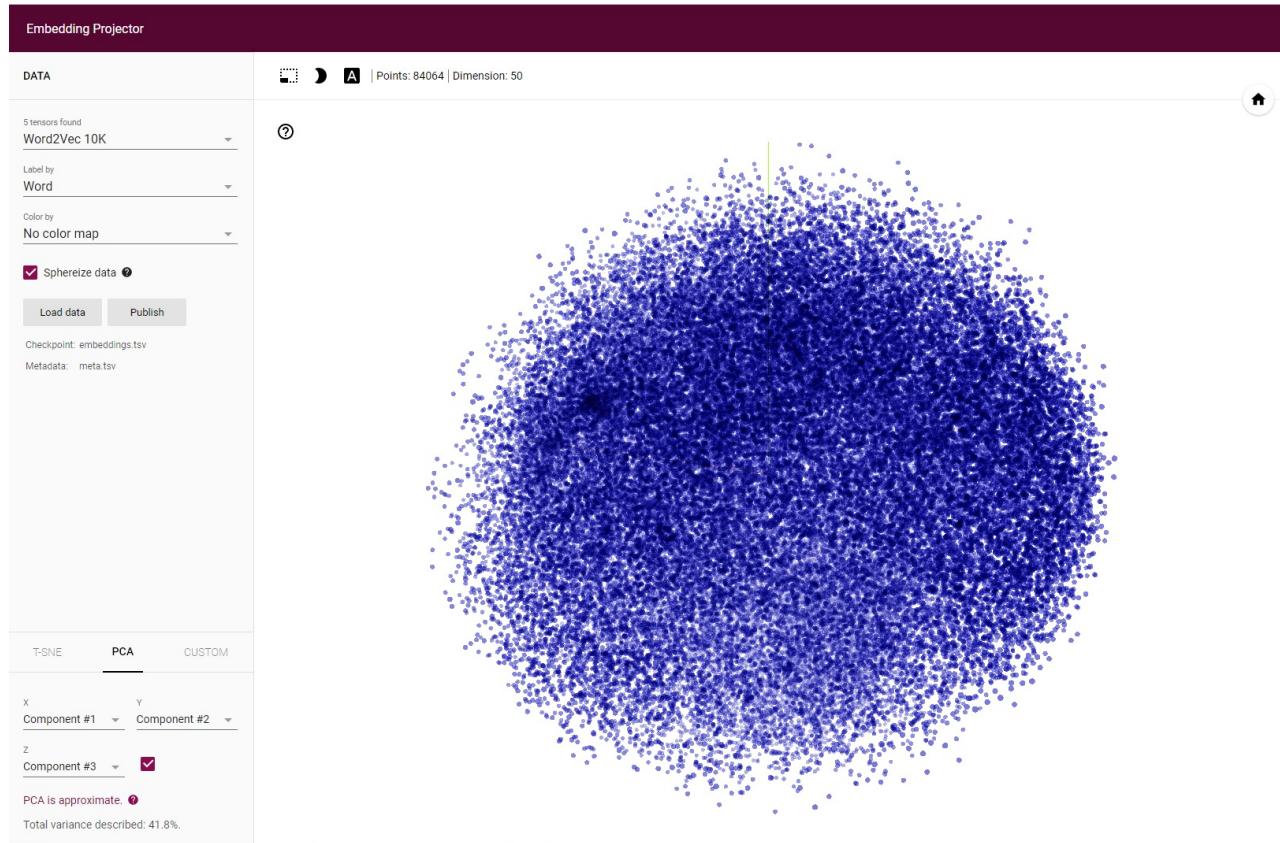
[Row(word=u'cerebrum'), Row(word=u'cerebellum'), Row(word=u'forebrain'), Row(word=u'neocortex'), Row(word=u'cerebrocortical'), Row(word=u'hippocampi'), Row(word=u'brains'), Row(word=u'white-matter'), Row(word=u'hippocampus'), Row(word=u'striatum'), Row(word=u'demyelinated'), Row(word=u'diencephalon'), Row(word=u'striatal'), Row(word=u'gerbil'), Row(word=u'infarcted'), Row(word=u'hypoxic-ischemic'), Row(word=u'retina'), Row(word=u'neurohypophysis'), Row(word=u'telencephalon'), Row(word=u'neocortical')]
```

We have shown two different ways of visualizing the embeddings. The first one uses a PCA to project the high dimensional vector to a 2-D vector space. This leads to a significant loss of information and the visualization is not as accurate. The second is to use PCA with **t-SNE**. t-SNE is a nonlinear dimensionality reduction technique that is well-suited for embedding high-dimensional data into a space of two or three dimensions, which can then be visualized in a scatter plot. It models each high-dimensional object by a two- or three-dimensional point in such a way that similar objects are modeled by nearby points and dissimilar objects are modeled by distant points. It works in two parts. First, it creates a probability distribution over the pairs in the higher dimensional space in a way that similar objects have a high probability of being picked and dissimilar points have low probability of getting picked. Second, it defines a similar probability distribution over the points in a low dimensional map and minimizes the KL Divergence between the two distributions with respect to location of points on the map. The location of the points in the low dimension is obtained by minimizing the KL Divergence using Gradient Descent. But t-SNE might

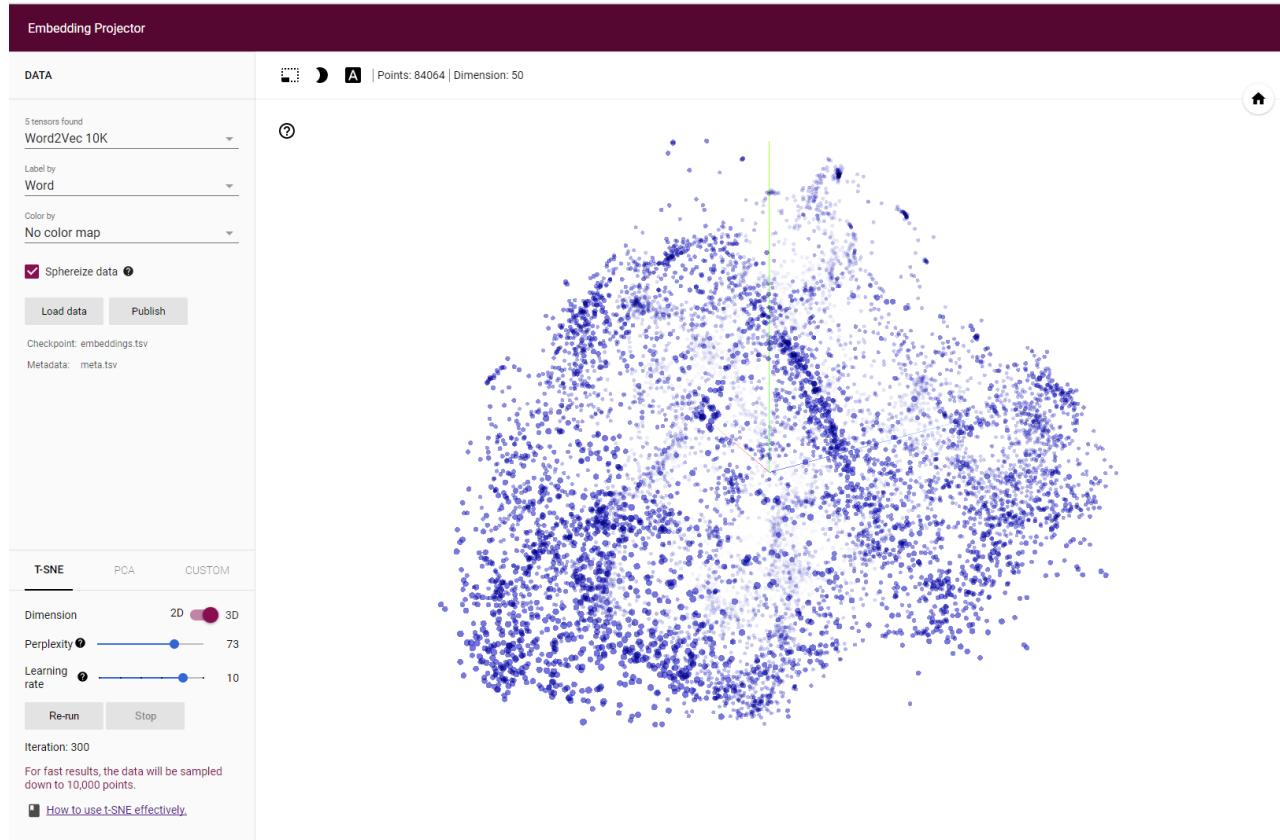
not be always reliable. Implementation details can be found [here](#).

As shown in the following figure, the t-SNE visualization provides more separation of word vectors and potential clustering patterns.

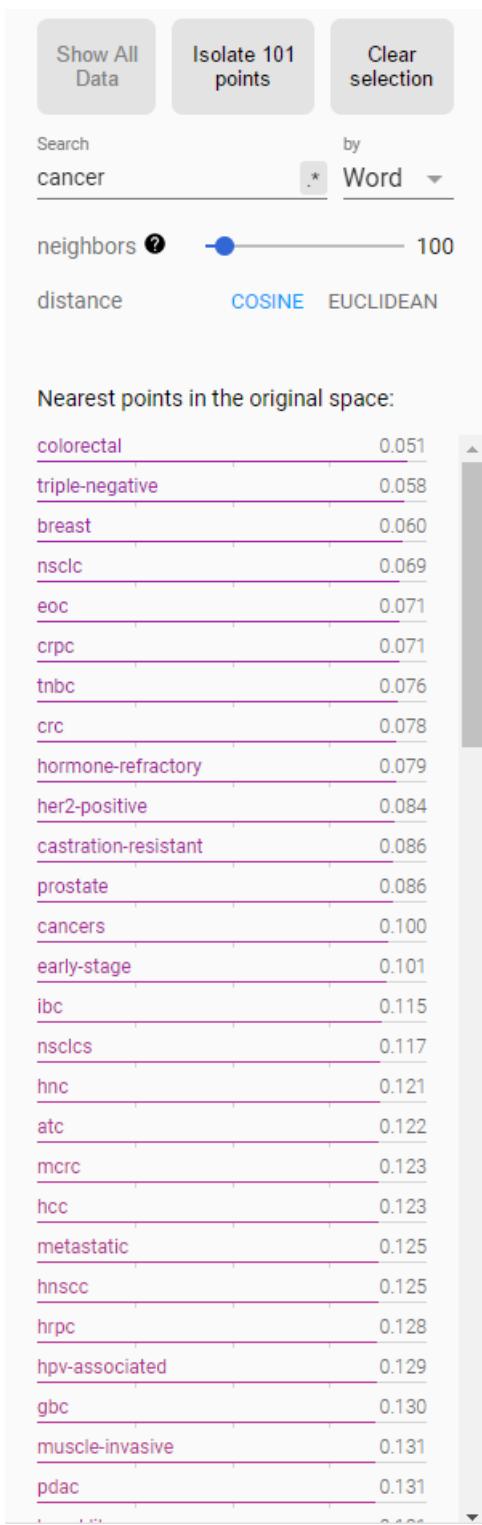
- Visualization with PCA



- Visualization with t-SNE



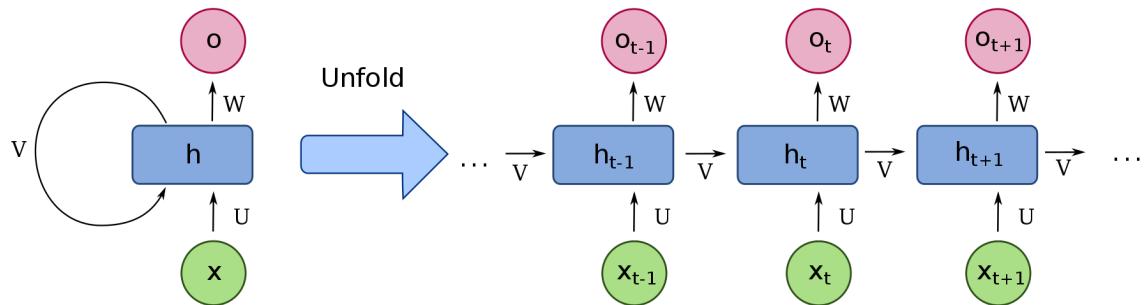
- Points closest to "Cancer" (they are all subtypes of Cancer)



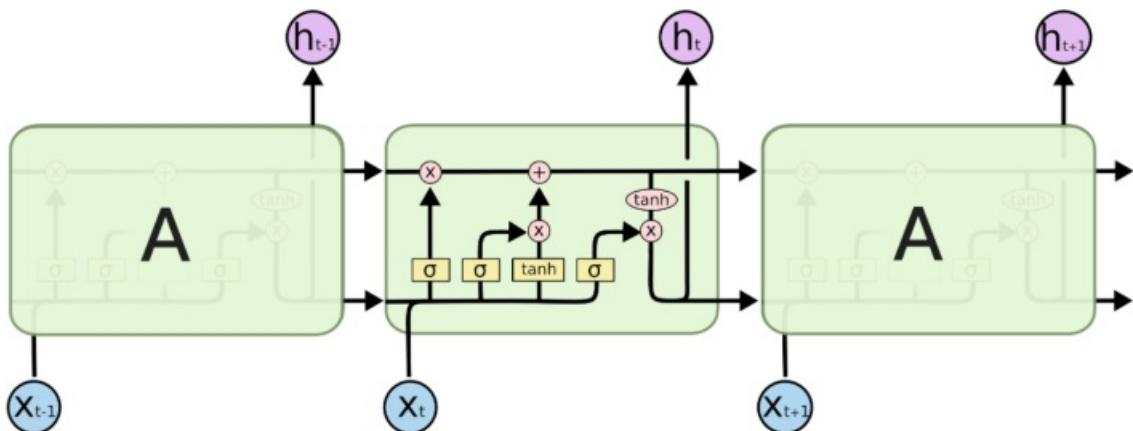
2.2. Train the neural entity extractor

See [Train the neural entity extractor](#).

The feed-forward neural network architecture suffers from a problem that they treat each input and output as independent of the other inputs and outputs. This architecture can't model sequence-to-sequence labeling tasks such as machine translation and entity extraction. Recurrent neural network models overcome this problem as they can pass information computed until now to the next node. This property is called having memory in the network since it is able to use the previously computed information as shown in the following figure:



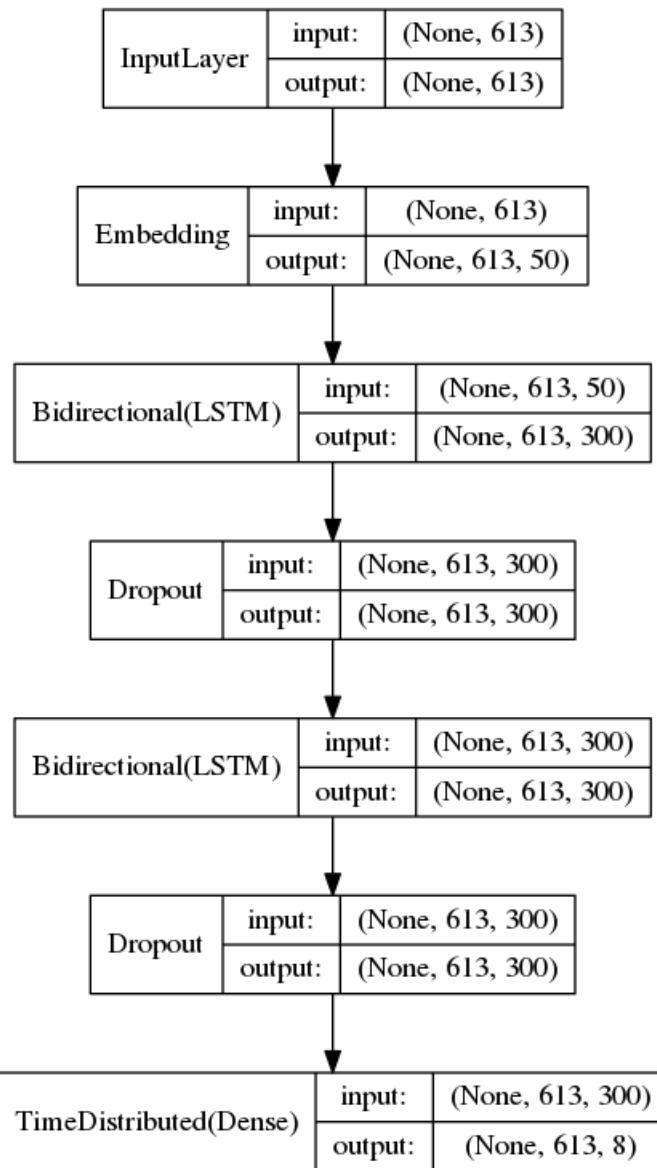
Vanilla RNNs actually suffer from the [Vanishing Gradient Problem](#) due to which they are not able to utilize all the information they have seen before. The problem becomes evident only when a large amount of context is required to make a prediction. But models like LSTM do not suffer from such a problem, in fact they are designed to remember long-term dependencies. Unlike vanilla RNNs that have a single neural network, the LSTMs have the interactions between four neural networks for each cell. For a detailed explanation of how LSTM work, refer to [this post](#).



The repeating module in an LSTM contains four interacting layers.

Let's try to put together our own LSTM-based recurrent neural network and try to extract entity types like drug, disease and symptom mentions from PubMed data. The first step is to obtain a large amount of labeled data and as you would have guessed, that's not easy! Most of the medical data contains lot of sensitive information about the person and hence are not publicly available. We rely on a combination of two different datasets that are publicly available. The first dataset is from Semeval 2013 - Task 9.1 (Drug Recognition) and the other is from BioCreative V CDR task. We are combining and auto labeling these two datasets so that we can detect both drugs and diseases from medical texts and evaluate our word embeddings. For implementation details, refer to [GitHub code link](#).

The model architecture that we have used across all the codes and for comparison is presented below. The parameter that changes for different datasets is the maximum sequence length (613 here).



2.3. Model evaluation

See [Model evaluation](#).

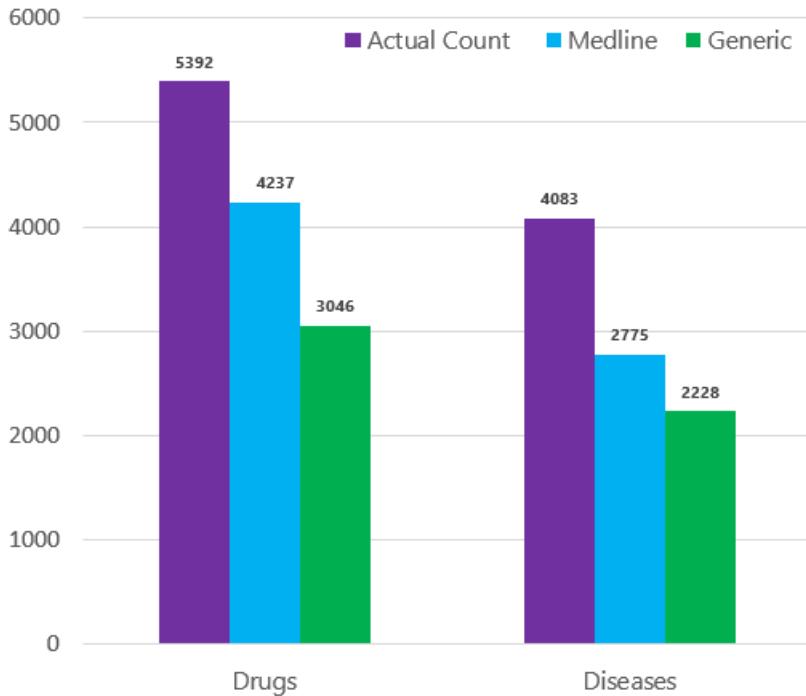
We use the evaluation script from the shared task [Bio-Entity Recognition Task at Bio NLP/NLPBA 2004](#) to evaluate the precision, recall, and F1 score of the model.

In-domain versus generic word embedding models

The following is a comparison between the accuracy of two feature types: (1) word embeddings trained on PubMed abstracts and (2) word embeddings trained on Google News. We clearly see that the in-domain model outperforms the generic model. Hence having a specific word embedding model rather than using a generic one is much more helpful.

- Task #1: Drugs and Diseases Detection

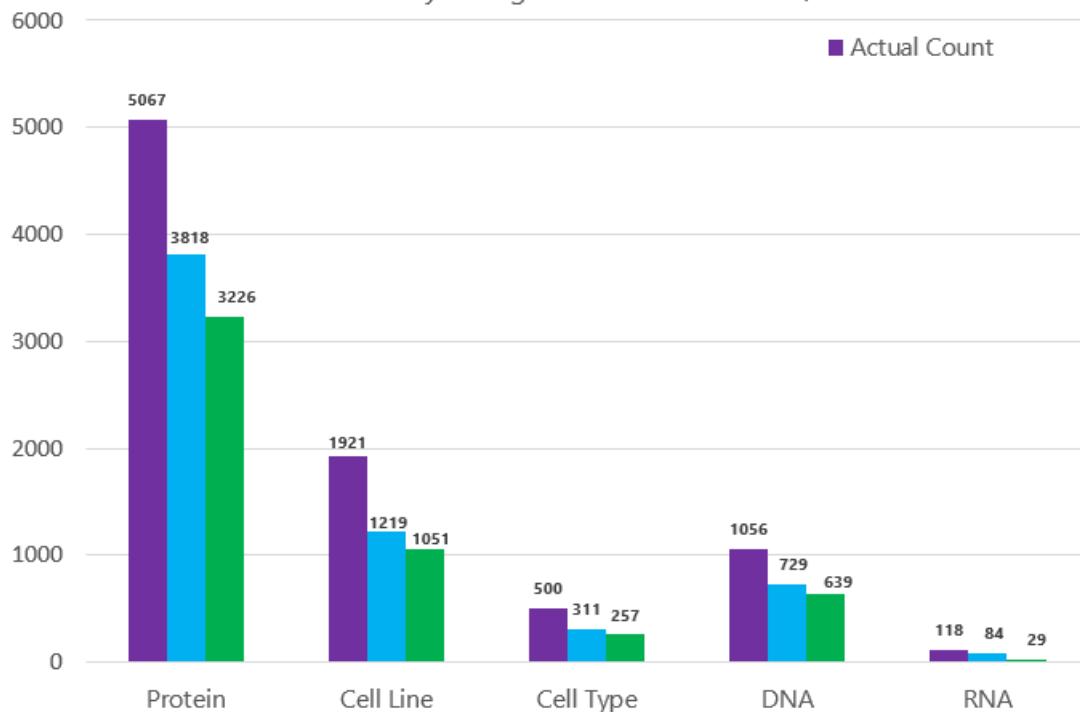
Drug and Disease dataset (auto-labeled)



We perform the evaluation of the word embeddings on other datasets in the similar fashion and see that in-domain model is always better.

- Task #2: Proteins, Cell Line, Cell Type, DNA, and RNA Detection

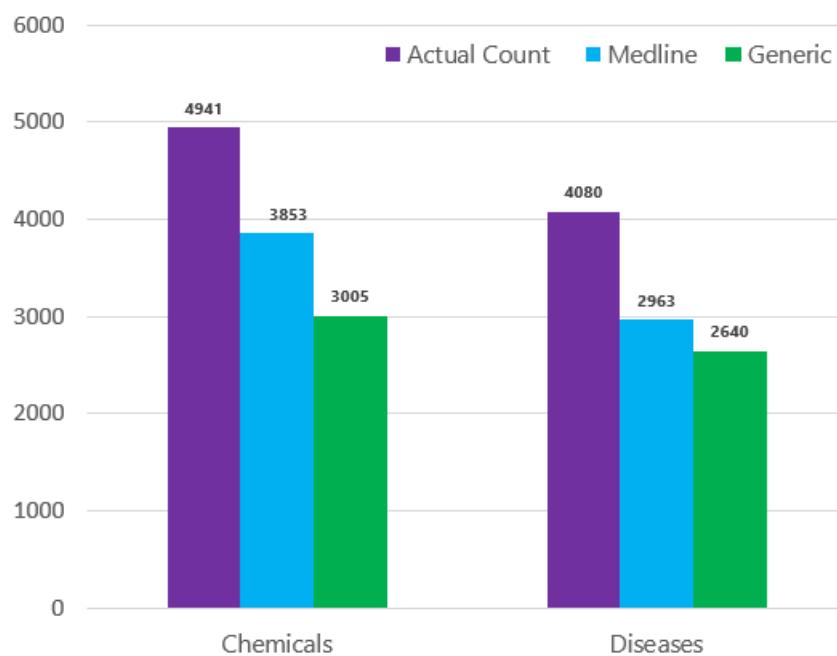
Bio-Entity Recognition Task at Bio NLP/NLPBA 2004



Embedding Dataset	Precision	Recall	F1 Score
MEDLINE	0.65	0.71	0.68
Google News	0.55	0.60	0.57

- Task #3: Chemicals and Diseases Detection

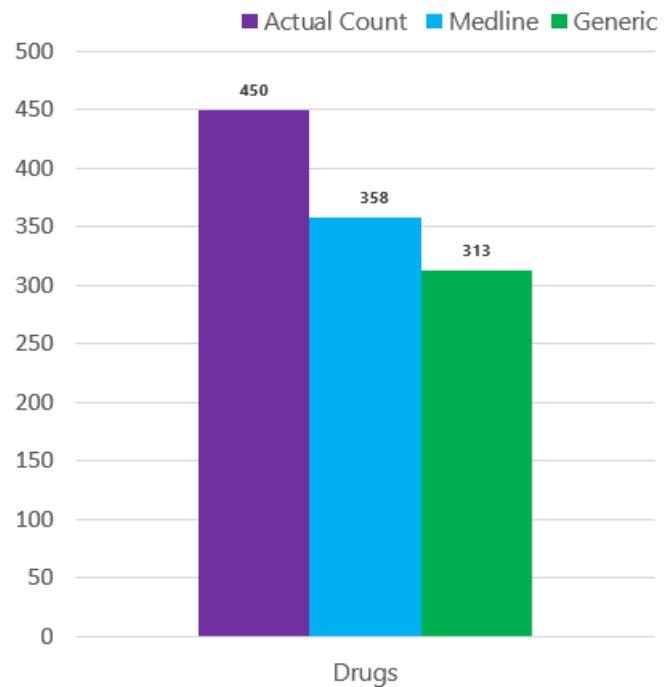
Bio Creative V CDR task corpus



Embedding Dataset	Precision	Recall	F1 Score
MEDLINE	0.77	0.76	0.76
Google News	0.74	0.63	0.67

- Task #4: Drugs Detection

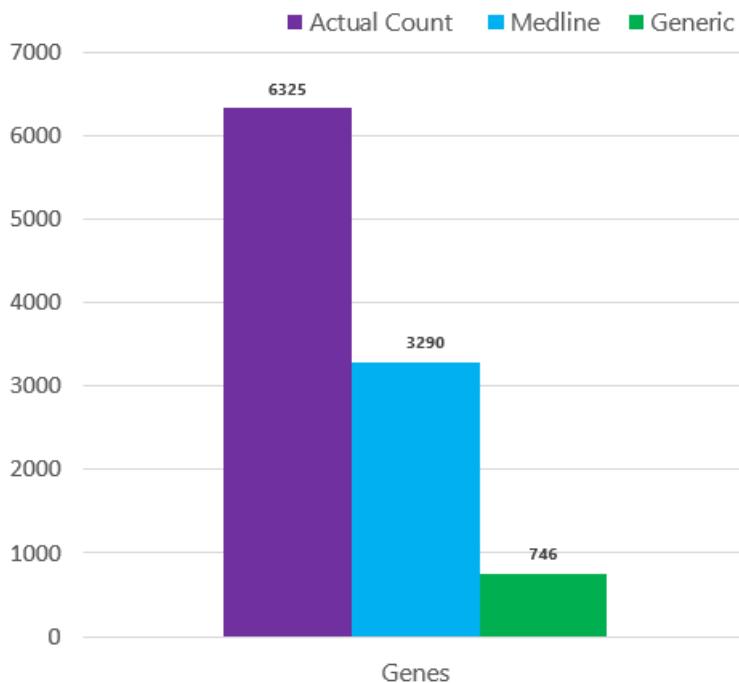
Semeval 2013 (Task 9.1 Drug Recognition)



Embedding Dataset	Precision	Recall	F1 Score
MEDLINE	0.93	0.79	0.85
Google News	0.92	0.69	0.79

- Task #5: Genes Detection

Bio creative 2 (GENE Recognition)

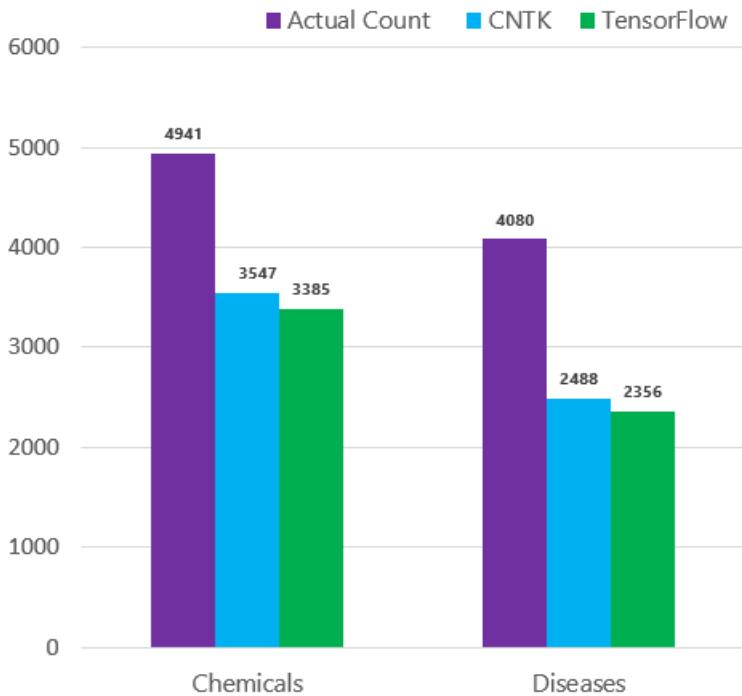


Embedding Dataset	Precision	Recall	F1 Score
MEDLINE	0.57	0.52	0.54
Google News	0.37	0.12	0.18

TensorFlow versus CNTK

All the reported models are trained using Keras with TensorFlow as backend. Keras with CNTK backend does not support "reverse" at the time this work was done. Therefore, for the sake of comparison, we have trained a unidirectional LSTM model with the CNTK backend and compared it to a unidirectional LSTM model with TensorFlow backend. Install CNTK 2.0 for Keras from [here](#).

Bio Creative V CDR task corpus



We concluded that CNTK performs as good as Tensorflow both in terms of the training time taken per epoch (60 secs for CNTK and 75 secs for Tensorflow) and the number of test entities detected. We are using the Unidirectional layers for evaluation.

3. Deployment

See [Deployment](#).

We deployed a web service on a cluster in the [Azure Container Service \(ACS\)](#). The operationalization environment provisions Docker and Kubernetes in the cluster to manage the web-service deployment. You can find further information about the operationalization process [here](#).

Conclusion

We went over the details of how you could train a word embedding model using Word2Vec algorithm on Spark and then use the extracted embeddings as features to train a deep neural network for entity extraction. We have applied the training pipeline on the biomedical domain. However, the pipeline is generic enough to be applied to detect custom entity types of any other domain. You just need enough data and you can easily adapt the workflow presented here for a different domain.

References

- Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. 2013a. Efficient estimation of word representations in vector space. In Proceedings of ICLR.
- Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg S Corrado, and Jeff Dean. 2013b. Distributed representations of words and phrases and their compositionality. In Proceedings of NIPS, pages 3111–3119.
- Billy Chiu, Gamal Crichton, Anna Korhonen, and Sampo Pyysalo. 2016. [How to Train Good Word Embeddings for Biomedical NLP](#), In Proceedings of the fifteenth Workshop on Biomedical Natural Language Processing, pages 166–174.

- Vector Representations of Words
- Recurrent Neural Networks
- Problems encountered with Spark ml Word2Vec
- Spark Word2Vec: lessons learned

Income classification with Team Data Science Process (TDSP) project

9/24/2018 • 7 minutes to read • [Edit Online](#)

NOTE

This article is deprecated. Support for earlier versions of this service will end incrementally. [View the support timeline](#). Start using the latest version with this [quickstart](#).

Introduction

Standardization of the structure and documentation of data science projects, that is anchored to an established [data science lifecycle](#), is key to facilitating effective collaboration in data science teams. Creating Azure Machine Learning projects with the [Team Data Science Process \(TDSP\)](#) template provides a framework for such standardization.

We had previously released a [GitHub repository for the TDSP project structure and templates](#). But it was not possible, until now to instantiate the TDSP structure and templates within a data science tool. We have now enabled creation of Azure Machine Learning projects that are instantiated with [TDSP structure and documentation templates for Azure Machine Learning](#). Instructions on how to use TDSP structure and templates in Azure Machine Learning is provided [here](#). Here we provide an example of how an actual machine learning project can be created using TDSP structure, populated with project-specific code, artifacts and documents, and executed within the Azure Machine Learning.

Link to GitHub repository

We provide summary documentation [here](#) about the sample. More extensive documentation can be found on the GitHub site.

Purpose

The primary purpose of this sample is to show how to instantiate and execute a machine learning project using the [Team Data Science Process \(TDSP\)](#) structure and templates in Azure Machine Learning. For this purpose, we use the well-known [1994 US Census data from the UCI Machine Learning Repository](#). The modeling task is to predict US annual income classes from US Census information (for example, age, race, education level, country of origin, etc.)

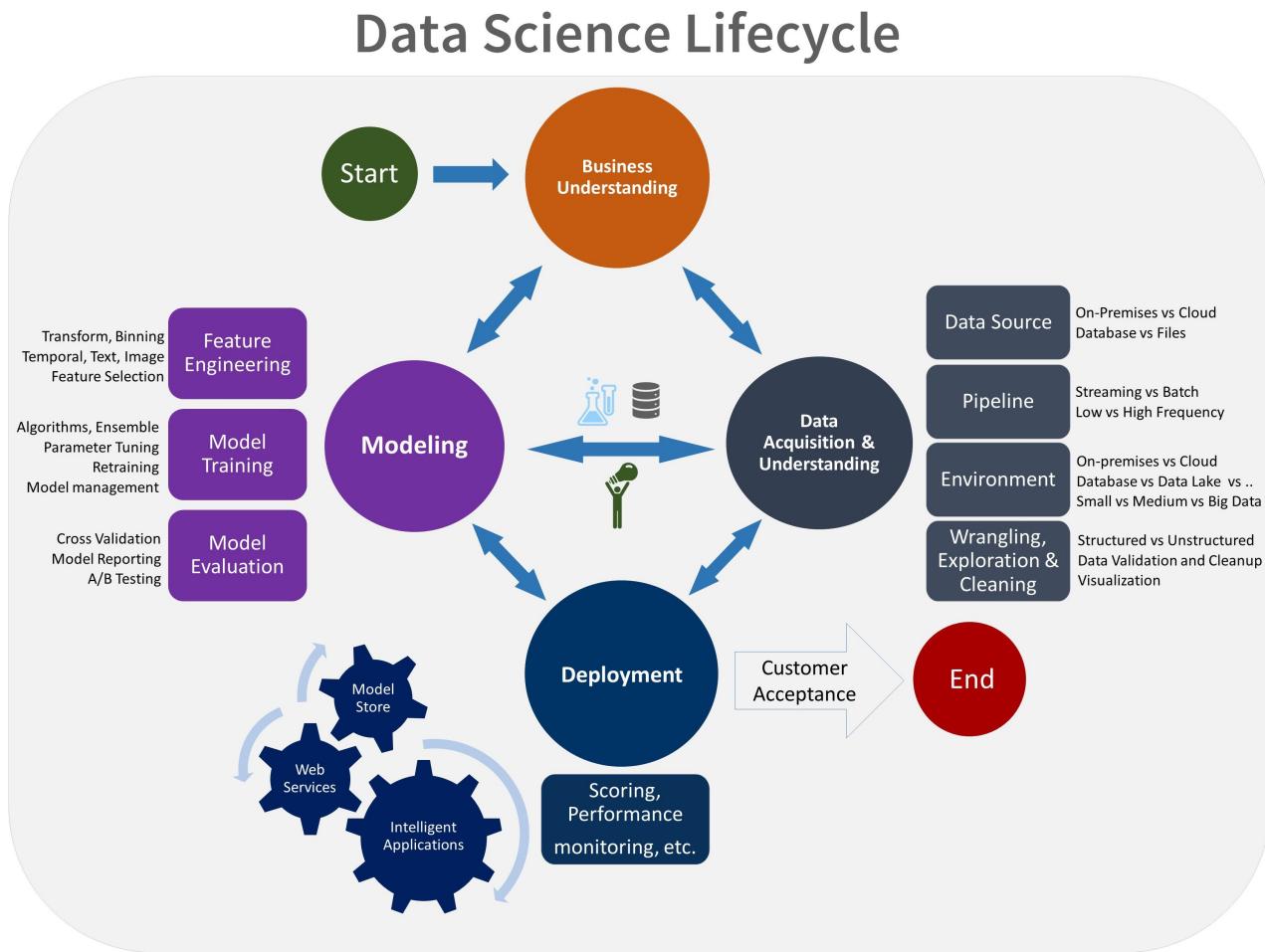
Scope

- Data exploration, training, and deployment of a machine learning model which address the prediction problem described in the Use Case Overview.
- Execution of the project in Azure Machine Learning using the Team Data Science Process (TDSP) template from Azure Machine Learning for this project. For project execution and reporting, we're going to use the TDSP lifecycle.
- Operationalization of the solution directly from Azure Machine Learning in Azure Container Services.

The project highlights several features of Azure Machine Learning, such TDSP structure instantiation and use, execution of code in Jupyter notebooks as well as Python files, and easy operationalization in Azure Container Services using Docker and Kubernetes.

Team Data Science Process (TDSP) lifecycle

See [Team Data Science Process \(TDSP\) Lifecycle](#)



Prerequisites

Required: subscription, hardware, software

1. An Azure [subscription](#). You can get a [free subscription](#) to execute this sample also.
2. An [Azure Data Science Virtual Machine \(DSVM\) Windows Server 2016](#), (VM Size: [DS3_V2](#), with 4 virtual CPUs and 14-Gb RAM). Although tested on an Azure DSVM, it is likely to work on any Windows 10 machine.
3. Review documentation on Azure Machine Learning and its related services (see below for links).
4. Make sure that you have properly installed Azure Machine Learning by the [quick start installation guide](#).

The dataset for this sample is from the UCI ML Repository [\[link\]](#). It is taken from the 1994 US Census database and contains census and income information for about 50,000 individuals. This is structured dataset having numerical and categorical features, and a categorical target consisting of two income categories ('> 50 K' or '<= 50 K').

Optional: Version control repository

If you would like to save and version your project and its contents, you need to have a version control repository where this can be done. You can enter the Git repository location while creating the new project using the TDSP template in Azure Machine Learning. See [how to use Git in Azure Machine Learning](#) for further details.

Informational: about Azure Machine Learning

- [FAQ - How to get started](#)
- [Overview](#)
- [Installation](#)
- [Execution](#)

- [Using TDSP](#)
- [Read and write files](#)
- [Using Git with Azure Machine Learning](#)
- [Deploying an ML model as a web service](#)

Create a new workbench project

Create a new project using this example as a template:

1. Open Azure Machine Learning Workbench
2. On the **Projects** page, click the + sign and select **New Project**
3. In the **Create New Project** pane, fill in the information for your new project
4. In the **Search Project Templates** search box, type "Classify US incomes - TDSP project" and select the template
5. Click **Create**

If you provide an empty Git repository location during creating the project (in the appropriate box), then that repository will be populated with the project structure and contents after creation of the project.

Use case overview

The problem is to understand how socio-economic data captured in US Census can help predict annual income of individuals in US. Based on such Census features, the machine learning task is to predict if the income of an individual is above \$50,000 or not (binary classification task).

Data description

For detailed information about the data, see the [description](#) in the UCI repository.

This data was extracted from the Census Bureau database found at: <https://www.census.gov/en.html>.

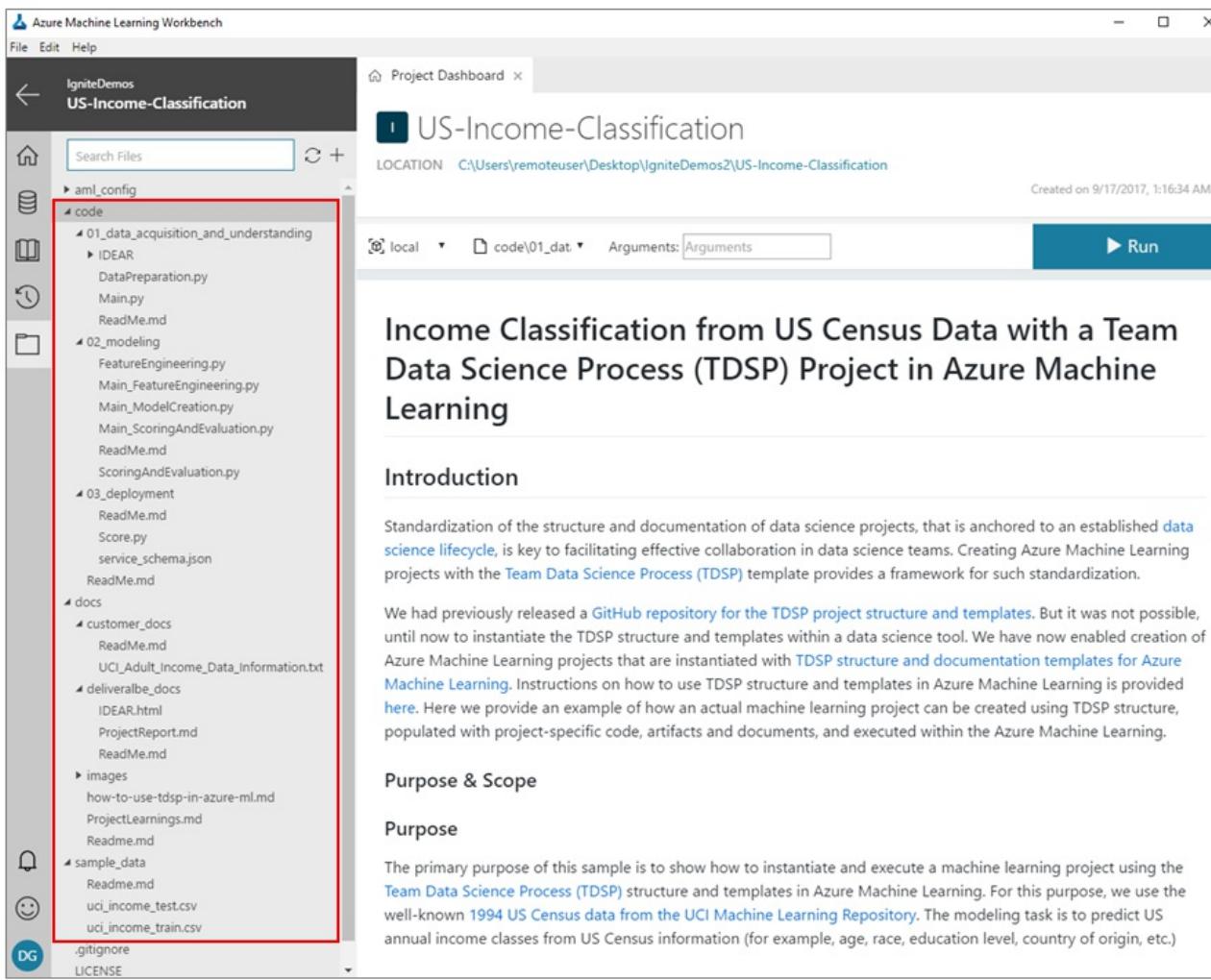
- There are a total of 48,842 instances (prior to any filtering), mix of continuous and discrete (train=32,561, test=16,281)
- Probability for the label '>50 K' : 23.93% / 24.78% (without unknowns)
- Probability for the label '<=50 K': 76.07% / 75.22% (without unknowns)
- **TARGET:** Income class '>50 K', '<=50 K'. These are replaced by 1 and 0 respectively in data preparation phase.
- **FEATURES:** Age, work class, education level, education level, race, sex, hours of work per week, etc.

Project structure, execution, and reporting

Structure

For this project, we use the TDSP folder structure and documentation templates (below), which follows the [TDSP lifecycle](#).

Project is created based on instructions provided [here](#). After it is filled with the project's code and artifacts, the structure looks as follows (see project structure boxed in red in figure below).



Execution

In this example, we execute code in **local compute environment**. Refer to Azure Machine Learning documents for further details on [execution options](#).

Executing a Python script in a local Python runtime is easy:

```
az ml experiment submit -c local my_script.py
```

IPython notebook files can be double-clicked from the project structure on the left of the Azure Machine Learning UI and run in the Jupyter Notebook Server.

The step-by-step data science workflow was as follows:

- **Data acquisition and understanding**

Data was downloaded in .csv form from URLs at UCI ML Repository [\[link\]](#). Features, target, and their transformations are described in detail in the ProjectReport.md file.

Code for data acquisition and understanding is located in: /code/01_data_acquisition_and_understanding.

Data exploration is performed using the Python 3 [IDEAR \(Interactive Data Exploration and Reporting\) utility](#) published as a part of [TDSP suite of data science tools](#). This utility helps to generate standardized data exploration reports for data containing numerical and categorical features and target. Details of how the Python 3 IDEAR utility was used is provided below.

The location of the final data exploration report is [IDEAR.html](#). A view of the IDEAR report is shown below:

Interactive Data Exploration, Analysis, and Reporting

- Author: Team Data Science Process from Microsoft
- Date: 2017/03
- Supported Data Sources: CSV files on the machine where the Jupyter notebook runs or data stored in SQL server
- Output: IDEAR_Report.ipynb

This is the **Interactive Data Exploration, Analysis and Reporting (IDEAR) in Python** running on Jupyter Notebook. The data can be stored in CSV file on the machine where the Jupyter notebook runs or from a query running against a SQL server. A yaml file has to be pre-configured before running this tool to provide information about the data.

Step 1: Configure and Set up IDEAR

Before start utilizing the functionalities provided by IDEAR, you need to first [configure and set up](#) the utilities by providing the yaml file and load necessary Python modules and libraries.

Step 2: Start using IDEAR

This tool provides various functionalities to help users explore the data and get insights through interactive visualization and statistical testing.

- [Read and Summarize the data](#)
- [Extract Descriptive Statistics of Data](#)
- [Explore Individual Variables](#)
- [Explore Interactions between Variables](#)
 - [Rank variables](#)
 - [Interaction between two categorical variables](#)
 - [Interaction between two numerical variables](#)
 - [Interaction between numerical and categorical variables](#)
 - [Interaction between two numerical variables and a categorical variable](#)
- [Visualize High Dimensional Data via Projecting to Lower Dimension Principal Component Spaces](#)
- [Generate Data Report](#)

After you are done with exploring the data interactively, you can choose to [show/hide the source code](#) to make your notebook look neater.

Note:

- Change the working directory and yaml file before running IDEAR in Jupyter Notebook.
- Run the cells and click **Export** button to export the code that generates the visualization/analysis result to temporary Jupyter notebooks.
- Run the last cell and click [Generate Final Report](#) to create `IDEAR_Report.ipynb` in the working directory. *If you do not export codes in some sections, you may see some warnings complaining that some temporary Jupyter Notebook files are missing.*
- Upload `IDEAR_Report.ipynb` to Jupyter Notebook server, and run it to generate report.

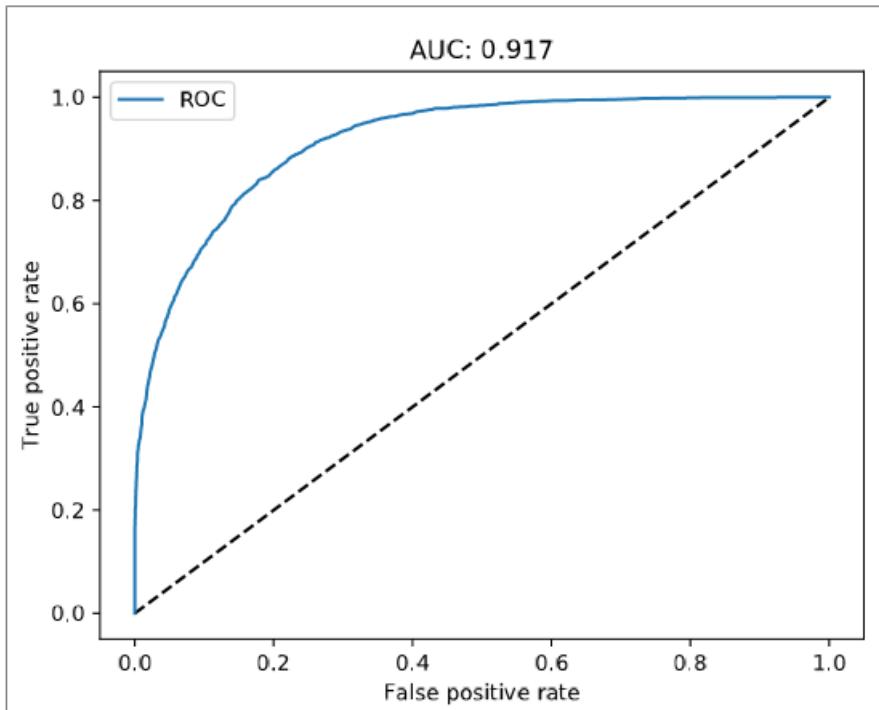
• Modeling

We created two models with 3-fold cross-validation: Elastic Net and Random forest. We used [59-point sampling](#) for random grid search as a strategy for cross-validation and model parameter optimization. Accuracy of the models were measured using AUC (Area under curve) on the test data set.

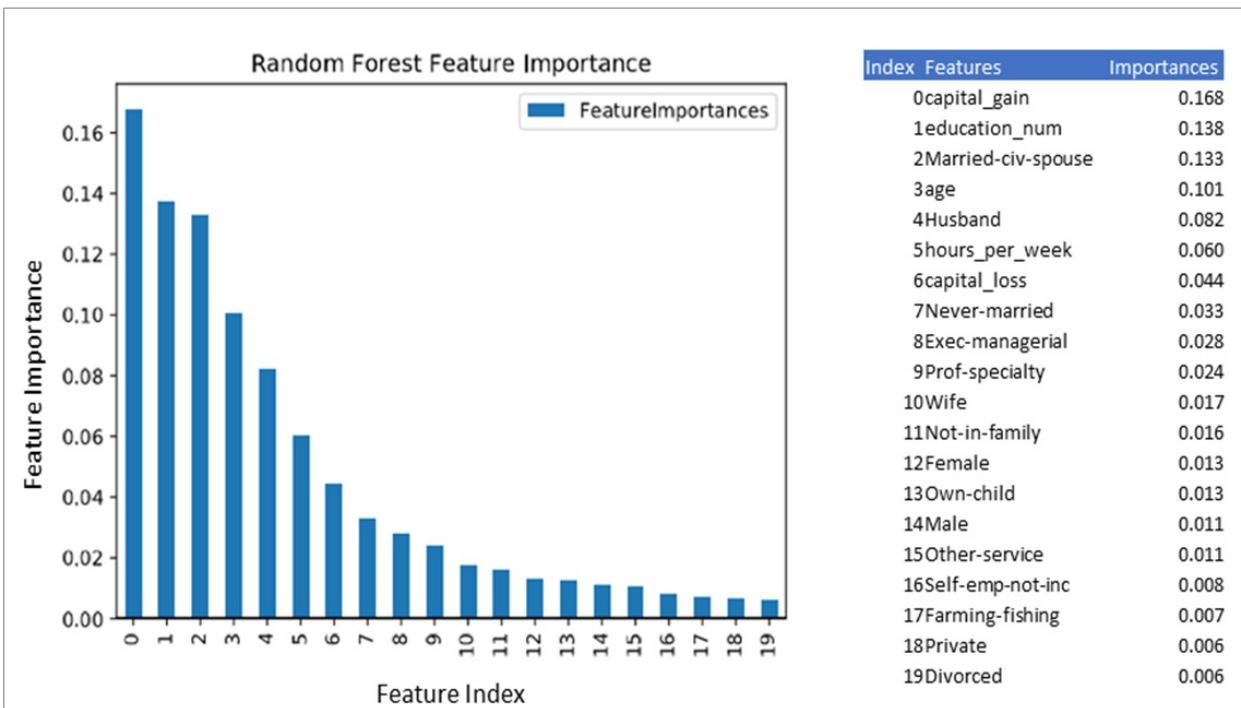
Code for modeling is located in: /code/02_modeling.

AUC of both Elastic Net and Random Forest models were > 0.85. We save both models in pickled.pkl files, and output the ROC plots for both models. AUC of Random Forest model was 0.92 and that of the Elastic Net model was 0.90. In addition, for model interpretation, feature importance for the Random Forest model are output in a .csv file and plotted in a pdf (top 20 predictive features only).

ROC curve of **Random Forest model** on test data is shown below. This was the model that was deployed:



Feature importance (top 20) of Random Forest model is shown below. It shows features capital gain amount, education, marital status, have highest feature importance.



• Deployment

We deployed the Random Forest model as a web-service on a cluster in the [Azure Container Service \(ACS\)](#). The operationalization environment provisions Docker and Kubernetes in the cluster to manage the web-service deployment. You can find further information on the operationalization process [here](#).

Code for deployment is located in: /code/03_deployment.

Final project report

Details about each of the above sections are provided in the compiled final project report [ProjectReport](#). The project report also contains further details about the use case, model performance metrics, deployment, and infrastructure on which the project was developed and deployed.

The project report, together with the contents of the entire project folder, and version control repository may be delivered to the client.

Conclusion

In this sample, we showed how to use TDSP structure and templates in Azure Machine Learning. Through the document and artifact templates you can:

1. Properly define purpose and scope of a project
2. Create a project team with distributed roles and responsibilities
3. Structure and execute projects according to the TDSP lifecycle stages
4. Develop standardized reports using TDSP data science utilities (such as the IDEAR data exploration and visualization report).
5. Prepare a final data science project report that can be delivered to a client

We hope you use this feature of Azure Machine Learning to facilitate with standardization and collaboration within your data science teams.

Next Steps

See references below to get started:

[How to use Team Data Science Process \(TDSP\) in Azure Machine Learning](#)

[Team Data Science Process \(TDSP\)](#)

[TDSP project template for Azure Machine Learning](#)

[US Income data-set from UCI ML repository](#)

Image classification using Azure Machine Learning Workbench

9/24/2018 • 18 minutes to read • [Edit Online](#)

NOTE

This article is deprecated. Support for earlier versions of this service will end incrementally. [View the support timeline](#). Start using the latest version with this [quickstart](#).

Image classification approaches can be used to solve a large number of Computer Vision problems. These include building models, which answer questions such as: *Is an OBJECT present in the image?* where OBJECT could for example be *dog, car, or ship*. Or more complex questions like: *What class of eye disease severity is evinced by this patient's retinal scan?*.

This tutorial addresses solving such problems. We show how to train, evaluate, and deploy your own image classification model using the [Microsoft Cognitive Toolkit \(CNTK\)](#) for deep learning. Example images are provided, but the reader can also bring their own dataset and train their own custom models.

Computer Vision solutions traditionally required expert knowledge to manually identify and implement so-called *features*, which highlight desired information in images. This manual approach changed in 2012 with the famous [AlexNet](#) [1] Deep Learning paper, and at present, Deep Neural Networks (DNN) are used to automatically find these features. DNNs led to a huge improvement in the field not just for image classification, but also for other Computer Vision problems such as object detection and image similarity.

Link to the gallery GitHub repository

<https://github.com/Azure/MachineLearningSamples-ImageClassificationUsingCNTK>

Overview

This tutorial is split into three parts:

- Part 1 shows how to train, evaluate, and deploy an image classification system using a pre-trained DNN as featurizer and training an SVM on its output.
- Part 2 then shows how to improve accuracy by, for example, refining the DNN rather than using it as a fixed featurizer.
- Part 3 covers how to use your own dataset instead of the provided example images, and if needed, how to produce your own dataset by scraping images from the net.

While previous experience with machine learning and CNTK is not required, it is helpful for understanding the underlying principles. Accuracy numbers, training time, etc. reported in the tutorial are only for reference, and the actual values when running the code almost certainly differ.

Prerequisites

The prerequisites to run this example are as follows:

1. An [Azure account](#) (free trials are available).
2. The [Azure Machine Learning Workbench](#) following the [quick start installation guide](#) to install the program and

- create a workspace.
3. A Windows machine. Windows OS is necessary since the Workbench supports only Windows and MacOS, while Microsoft's Cognitive Toolkit (which we use as deep learning library) only supports Windows and Linux.
 4. A dedicated GPU is not required to execute the SVM training in part 1, however it is needed for refining of the DNN described in part 2. If you lack a strong GPU, want to train on multiple GPUs, or do not have a Windows machine, then consider using Azure's Deep Learning Virtual Machine with Windows operating system. See [here](#) for a 1-click deployment guide. Once deployed, connect to the VM via a remote desktop connection, install Workbench there, and execute the code locally from the VM.
 5. Various Python libraries such as OpenCV need to be installed. Click *Open Command Prompt* from the *File* menu in the Workbench and run the following commands to install these dependencies:
 - `pip install https://cntk.ai/PythonWheel/GPU/cntk-2.2-cp35-cp35m-win_amd64.whl`
 - `pip install opencv_python-3.3.1-cp35-cp35m-win_amd64.whl` after downloading the OpenCV wheel from <http://www.lfd.uci.edu/~gohlke/pythonlibs/> (the exact filename and version can change)
 - `conda install pillow`
 - `pip install -U numpy`
 - `pip install bqplot`
 - `jupyter nbextension enable --py --sys-prefix bqplot`
 - `jupyter nbextension enable --py widgetsnbextension`

Troubleshooting / Known bugs

- A GPU is needed for part 2, and otherwise the error "Batch normalization training on CPU is not yet implemented" is thrown when trying to refine the DNN.
- Out-of-memory errors during DNN training can be avoided by reducing the minibatch size (variable `cntk_mb_size` in `PARAMETERS.py`).
- The code was tested using CNTK 2.2, and should run also on older (up to v2.0) and newer versions without any or only minor changes.
- At the time of writing, the Azure Machine Learning Workbench had problems showing notebooks larger than 5 Mbytes. Notebooks of this large size can happen if the notebook is saved with all cell output displayed. If you encounter this error, then open the command prompt from the File menu inside the Workbench, execute `jupyter notebook`, open the notebook, clear all output, and save the notebook. After performing these steps, the notebook will open properly inside the Azure Machine Learning Workbench again.
- All scripts provided in this sample have to be executed locally, and not on e.g. a docker remote environment. All notebooks need to be executed with kernel set to the local project kernel with name "PROJECTNAME local" (e.g. "myImgClassUsingCntk local").

Create a new workbench project

To create a new project using this example as a template:

1. Open Azure Machine Learning Workbench.
2. On the **Projects** page, click the + sign and select **New Project**.
3. In the **Create New Project** pane, fill in the information for your new project.
4. In the **Search Project Templates** search box, type "Image classification" and select the template.
5. Click **Create**.

Performing these steps creates the project structure shown below. The project directory is restricted to be less than 25 Mbytes since the Azure Machine Learning Workbench creates a copy of this folder after each run (to enable run history). Hence, all image and temporary files are saved to and from the directory `~/Desktop/imgClassificationUsingCntk_data` (referred to as *DATA_DIR* in this document).

FOLDER	DESCRIPTION
aml_config/	Directory containing the Azure Machine Learning Workbench configuration files
libraries/	Directory containing all Python and Jupyter helper functions
notebooks/	Directory containing all notebooks
resources/	Directory containing all resources (for example url of fashion images)
scripts/	Directory containing all scripts
PARAMETERS.py	Python script specifying all parameters
readme.md	This readme document

Data description

This tutorial uses as running example an upper body clothing texture dataset consisting of up to 428 images. Each image is annotated as one of three different textures (dotted, striped, leopard). We kept the number of images small so that this tutorial can be executed quickly. However, the code is well-tested and works with tens of thousands of images or more. All images were hand-annotated as is explained in [Part 3](#). The image URLs with their respective attributes are listed in the `/resources/fashionTextureUrls.tsv` file.

The script `0_downloadData.py` downloads all images to the `DATA_DIR/images/fashionTexture/` directory. Some of the 428 URLs are likely broken. This is not an issue, and just means that we have slightly fewer images for training and testing. All scripts provided in this sample have to be executed locally, and not on e.g. a docker remote environment.

The following figure shows examples for the attributes dotted (left), striped (middle), and leopard (right). Annotations were done according to the upper body clothing item.



Part 1 - Model training and evaluation

In the first part of this tutorial, we are training a system that uses, but does not modify, a pre-trained deep neural network. This pre-trained DNN is used as a featurizer, and a linear SVM is trained to predict the attribute (dotted, striped, or leopard) of a given image.

We now described this approach in detail, step-by-step, and show which scripts need to be executed. We recommend after each step to inspect which files are written and where they are written to.

All important parameters are specified, and a short explanation provided, in a single place: the `PARAMETERS.py` file.

Step 1: Data preparation

Script: `1_prepareData.py`. Notebook: `showImages.ipynb`

The notebook `showImages.ipynb` can be used to visualize the images, and to correct their annotation as needed. To run the notebook, open it in Azure Machine Learning Workbench, click on "Start Notebook Server" if this option is shown, change to the local project kernel with name "PROJECTNAME local" (e.g. "myImgClassUsingCntk local"), and then execute all cells in the notebook. See the troubleshooting section in this document if you get an error complaining that the notebook is too large to be displayed.



Now execute the script named `1_prepareData.py`, which assigns all images to either the training set or the test set. This assignment is mutually exclusive - no training image is also used for testing or vice versa. By default, a random 75% of the images from each attribute class are assigned to training, and the remaining 25% are assigned to testing. All data generated by the script are saved in the `DATA_DIR/proc/fashionTexture/` folder.

```
PARAMETERS: datasetName = fashionTexture
Directory used to read and write model/image files: C:\Users\pabuehle\Desktop\imgClassificationUsingCntk_data/
Split images into train or test...
    Training: 89 images in directory dotted
    Testing: 30 images in directory dotted
    Training: 73 images in directory leopard
    Testing: 25 images in directory leopard
    Training: 123 images in directory striped
    Testing: 42 images in directory striped
DONE.
```

Step 2: Refining the Deep Neural Network

Script: `2_refineDNN.py`

As we explained in part 1 of this tutorial, the pre-trained DNN is kept fixed (that is, it is not refined). However, the script named `2_refineDNN.py` is still executed in part 1, as it loads a pre-trained [ResNet](#) [2] model and modifies it, for example, to allow for higher input image resolution. This step is fast (seconds) and does not require a GPU.

In part 2 of the tutorial, a modification to the `PARAMETERS.py` file causes the `2_refineDNN.py` script to also refine the pre-trained DNN. By default, we run 45 training epochs during refinement.

In both cases, the final model is then written to the file `DATA_DIR/proc/fashionTexture/cntk_fixed.model`.

Step 3: Evaluate DNN for all images

Script: `3_runDNN.py`

We can now use the (possibly refined) DNN from the last step to featurize our images. Given an image as input to the DNN, the output is the 512-floats vector from the penultimate layer of the model. This vector is much smaller dimensional than the image itself. Nevertheless, it should contain (and even highlight) all information in the image relevant to recognize the image's attribute, that is, if the clothing item has a dotted, striped, or leopard texture.

All of the DNN image representations are saved to the file
DATA_DIR/proc/fashionTexture/cntkFiles/features.pickle.

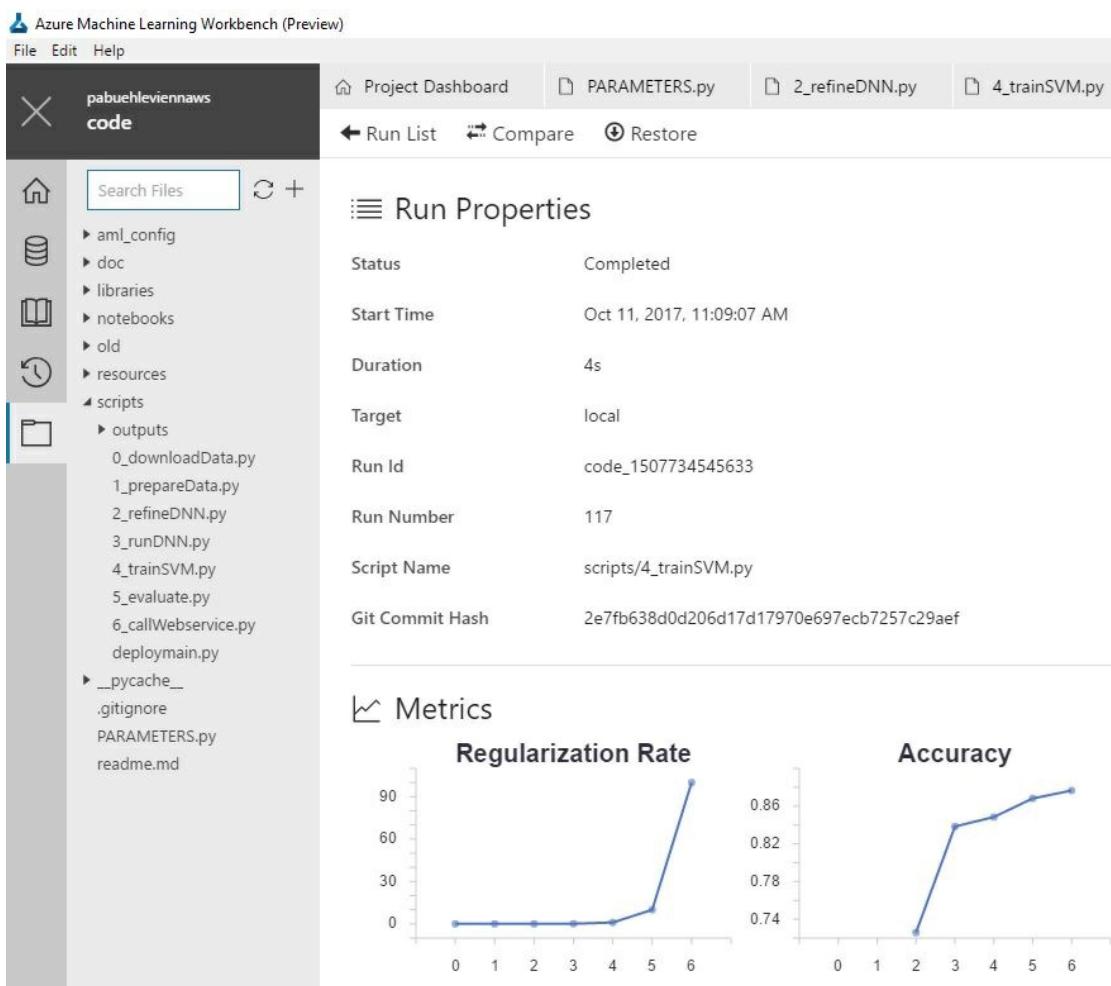
```
PARAMETERS: datasetName = fashionTexture
Using GPU for CNTK training/scoring.
Directory used to read and write model/image files: C:\Users\pabuehle\Desktop\imgClassificationUsingCntk_data\
Running DNN for training set..
Evaluating DNN (output dimension = 512) for image 128 of 285: C:\Users\pabuehle\Desktop\imgClassificationUsingCntk_data\images\fashionTexture\striped\408.jpg
Evaluating DNN (output dimension = 512) for image 256 of 285: C:\Users\pabuehle\Desktop\imgClassificationUsingCntk_data\images\fashionTexture\leopard\201.jpg
Running DNN for test set..
Writing CNTK outputs to file C:\Users\pabuehle\Desktop\imgClassificationUsingCntk_data\proc\fashionTexture\features_svm.pickle ...
DONE.
```

Step 4: Support Vector Machine training

Script: 4_trainSVM.py

The 512-floats representations computed in the last step are now used to train an SVM classifier: given an image as input, the SVM outputs a score for each attribute to be present. In our example dataset, this means a score for 'striped', for 'dotted', and for 'leopard'.

Script 4_trainSVM.py loads the training images, trains an SVM for different values of the regularization (slack) parameter C, and keeps the SVM with highest accuracy. The classification accuracy is printed on the console and plotted in the Workbench. For the provided texture data these values should be around 100% and 88% respectively. Finally, the trained SVM is written to the file *DATA_DIR/proc/fashionTexture/cntkFiles/svm.np*.



Step 5: Evaluation and visualization

Script: 5_evaluate.py. Notebook: showResults.ipynb

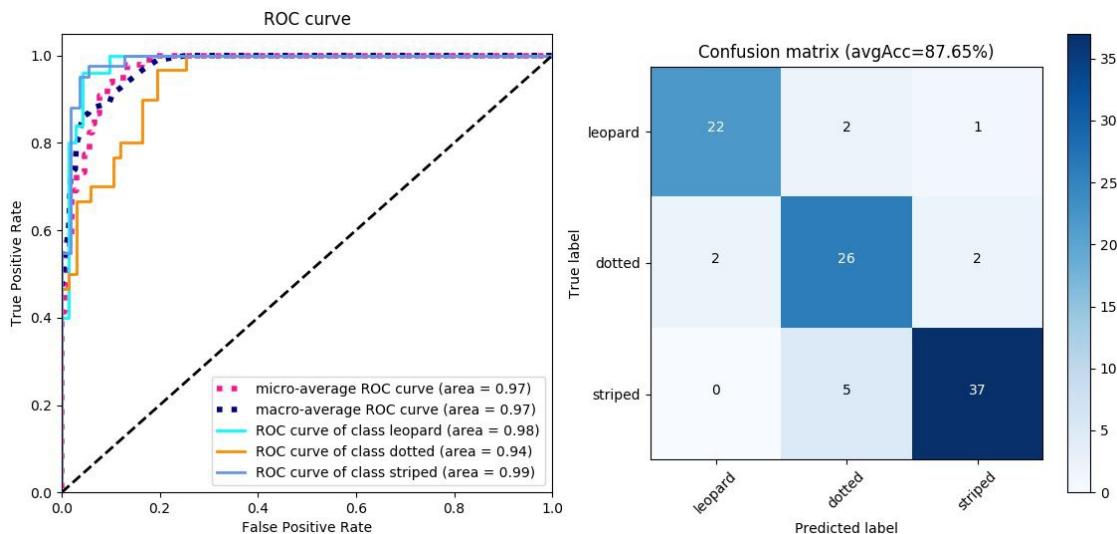
The accuracy of the trained image classifier can be measured using the script 5_evaluate.py. The script scores all

test images using the trained SVM classifier, assigns each image the attribute with the highest score, and compares the predicted attributes with the ground truth annotations.

The output of script `5_evaluate.py` is shown below. The classification accuracy of each individual class is computed, as well as the accuracy for the full test set ('overall accuracy'), and the average over the individual accuracies ('overall class-averaged accuracy'). 100% corresponds to the best possible accuracy and 0% to the worst. Random guessing would on average produce a class-averaged accuracy of 1 over the number of attributes: in our case, this accuracy would be 33.33%. These results improve significantly when using a higher input resolution such as `rf_inputResoluton = 1000`, however at the expense of longer DNN computation times.

```
PARAMETERS: datasetName = fashionTexture
Classifier = svm
Directory used to read and write model/image files: C:\Users\pabuehle\Desktop/imgClassificationUsingCntrk_data/
Loading data...
Evaluate SVM...
Class leopard accuracy: 88.00%.
Class dotted accuracy: 86.67%.
Class striped accuracy: 88.10%.
OVERALL accuracy: 87.63%.
OVERALL class-averaged accuracy: 87.59%.
DONE.
```

In addition to accuracy, the ROC curve is plotted with respective area-under-curve (left); and the confusion matrix is shown (right):



Finally, the notebook `showResults.py` is provided to scroll through the test images and visualize their respective classification scores. As explained in step1, every notebook in this sample needs to use the local project kernel with name "PROJECTNAME local":



Step 6: Deployment

Scripts: `6_callWebservice.py`, `deploymain.py`. Notebook: `deploy.ipynb`

The trained system can now be published as a REST API. Deployment is explained in the notebook `deploy.ipynb`, and based on functionality within the Azure Machine Learning Workbench (remember to set as kernel the local project kernel with name "PROJECTNAME local"). See also the excellent deployment section of the [IRIS tutorial](#) for more deployment related information.

Once deployed, the web service can be called using the script `6_callWebservice.py`. Note that the IP address (either local or on the cloud) of the web service needs to be set first in the script. The notebook `deploy.ipynb` explains how to find this IP address.

Part 2 - Accuracy improvements

In part 1, we showed how to classify an image by training a linear Support Vector Machine on the 512-floats output of a Deep Neural Network. This DNN was pre-trained on millions of images, and the penultimate layer returned as feature vector. This approach is fast since the DNN is used as-is, but nevertheless often gives good results.

We now present several ways to improve the accuracy of the model from part 1. Most notably we refine the DNN rather than keeping it fixed.

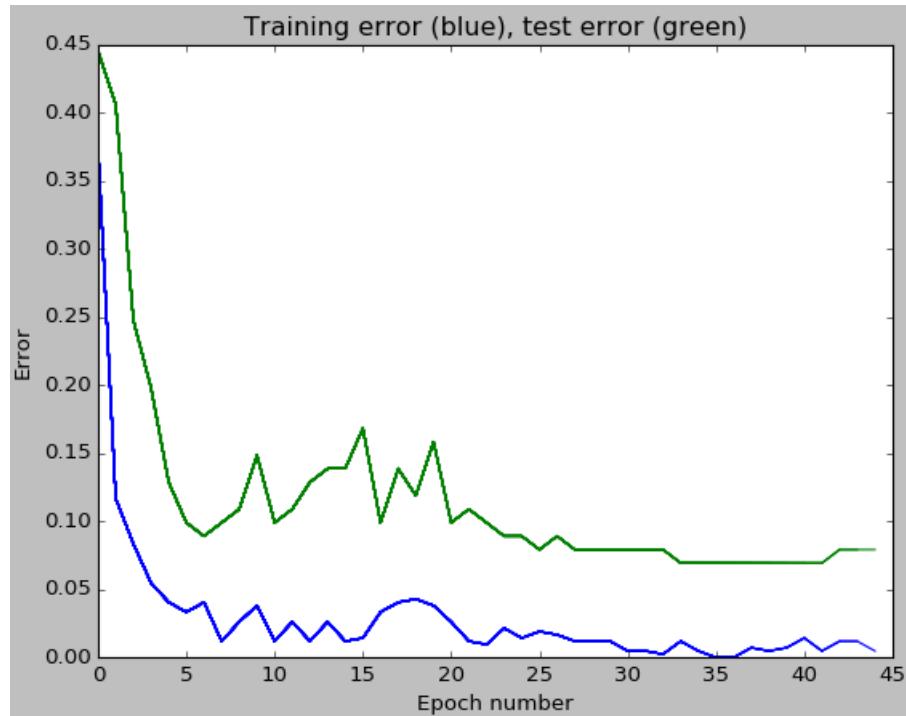
DNN refinement

Instead of an SVM, one can do the classification directly in the neural network. This is achieved by adding a new last layer to the pre-trained DNN, which takes the 512-floats from the penultimate layer as input. The advantage of doing the classification in the DNN is that now the full network can be retrained using backpropagation. This approach often leads to much better classification accuracies compared to using the pre-trained DNN as-is, however at the expense of much longer training time (even with GPU).

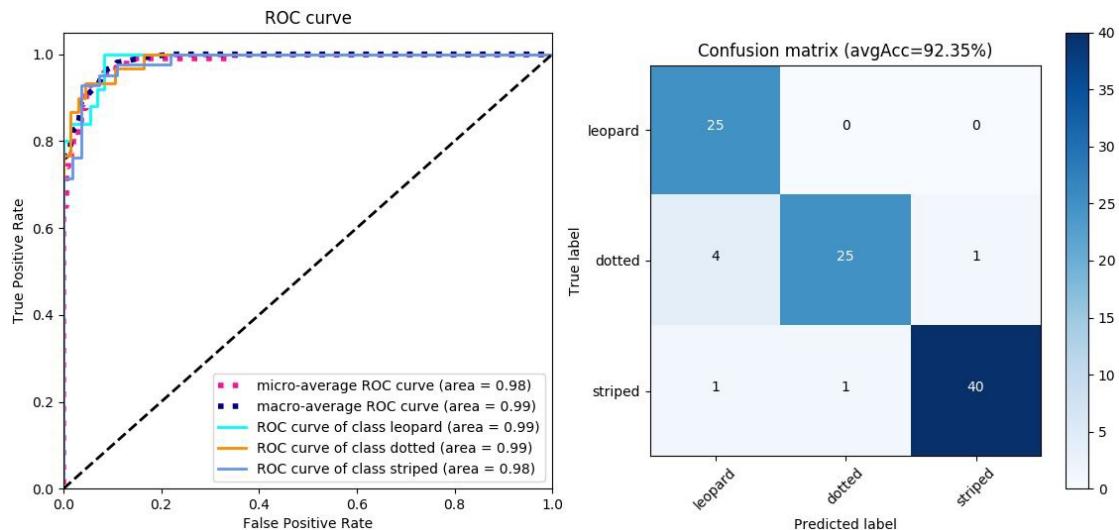
Training the Neural Network instead of an SVM is done by changing the variable `classifier` in `PARAMETERS.py` from `svm` to `dnn`. Then, as described in part 1, all the scripts except for data preparation (step 1) and SVM training (step 4) need to be executed again. DNN refinement requires a GPU if no GPU was found or if the GPU is locked (for example by a previous CNTK run) then script `2_refineDNN.py` throws an error. DNN training can throw

out-of-memory error on some GPUs, which can be avoided by reducing the minibatch size (variable `cntk_mb_size` in `PARAMETERS.py`).

Once training completes, the refined model is saved to `DATA_DIR/proc/fashionTexture/cntk_refined.model`, and a plot drawn which shows how the training and test classification errors change during training. Note in that plot that the error on the training set is much smaller than on the test set. This so-called over-fitting behavior can be reduced, for example, by using a higher value for the dropout rate `rf_dropoutRate`.



As can be seen in the plot below, the accuracy using DNN refinement on the provided dataset is 92.35% versus the 88.92% before (part 1). In particular, the 'dotted' images improve significantly, with an ROC area-under-curve of 0.98 with refinement vs. 0.94 before. We are using a small dataset, and hence the actual accuracies running the code are different. This discrepancy is due to stochastic effects such as the random split of the images into training and testing sets.

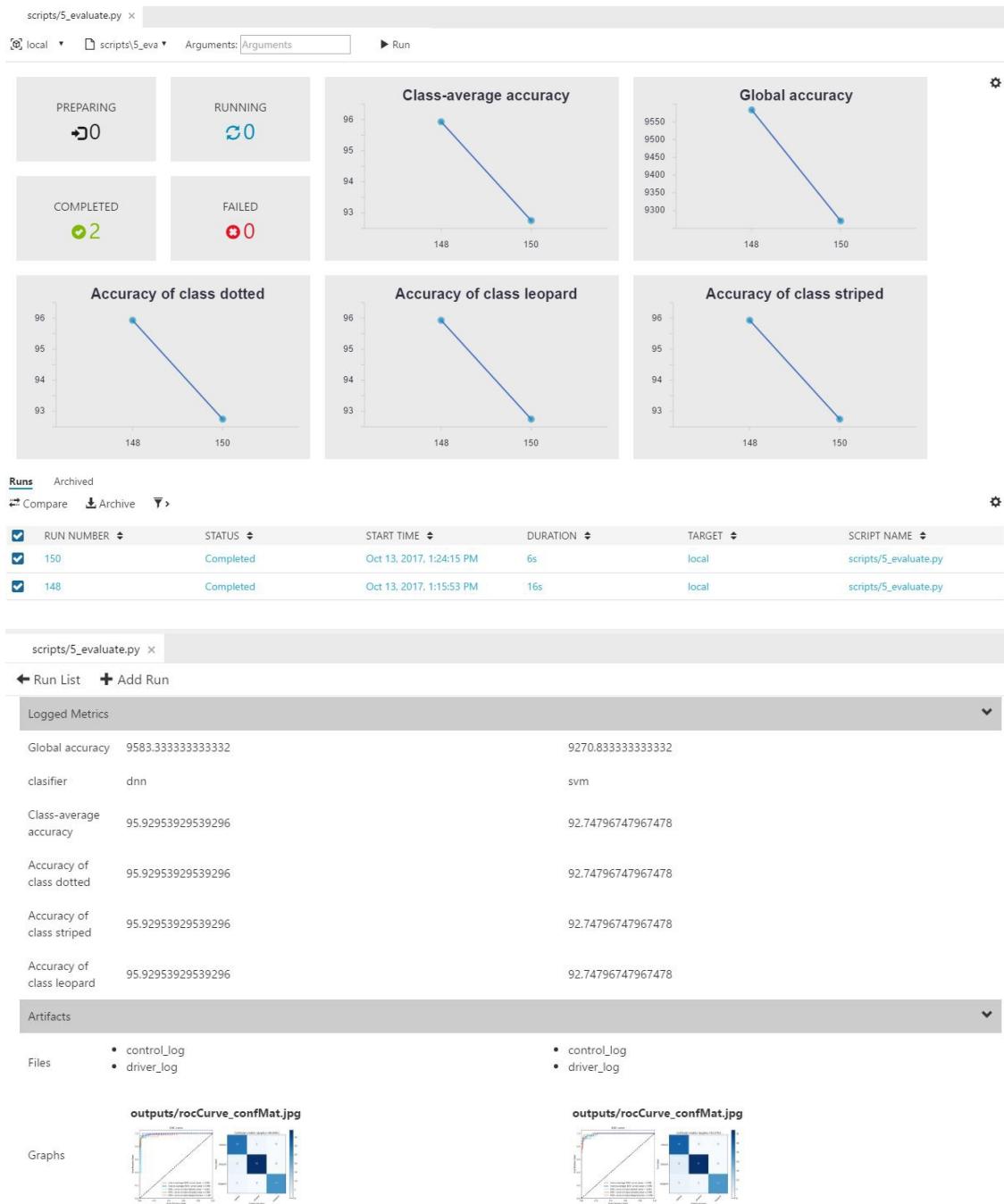


Run history tracking

The Azure Machine Learning Workbench stores the history of each run on Azure to allow comparison of two or more runs that are even weeks apart. This is explained in detail in the [Iris tutorial](#). It is also illustrated in the following screenshots where we compare two runs of the script `5_evaluate.py`, using either DNN refinement that is, `classifier = "dnn"` (run number 148) or SVM training that is, `classifier = "svm"` (run number 150).

In the first screenshot, the DNN refinement leads to better accuracies than SVM training for all classes. The second

screenshot shows all metrics that are being tracked, including what the classifier was. This tracking is done in the script `5_evaluate.py` by calling the Azure Machine Learning Workbench logger. In addition, the script also saves the ROC curve and confusion matrix to the *outputs* folder. This *outputs* folder is special in that its content is also tracked by the Workbench history feature and hence the output files can be accessed at any time, regardless of whether local copies have been overwritten.



Parameter tuning

As is true for most machine learning projects, getting good results for a new dataset requires careful parameter tuning as well as evaluating different design decisions. To help with these tasks, all important parameters are specified, and a short explanation provided, in a single place: the `PARAMETERS.py` file.

Some of the most promising avenues for improvements are:

- **Data quality:** Ensure the training and test sets have high quality. That is, the images are annotated correctly, ambiguous images removed (for example clothing items with both stripes and dots), and the attributes are mutually exclusive (that is, chosen such that each image belongs to exactly one attribute).
- If the object-of-interest is small in the image then Image classification approaches are known not to work well. In such cases consider using an object detection approach as described in this [tutorial](#).

- DNN refinement: The arguably most important parameter to get right is the learning rate `rf_lrPerMb`. If the accuracy on the training set (first figure in part 2) is not close to 0-5%, most likely it is due to a wrong learning rate. The other parameters starting with `rf_` are less important. Typically, the training error should decrement exponentially and be close to 0% after training.
- Input resolution: The default image resolution is 224x224 pixels. Using higher image resolution (parameter: `rf_inputResolution`) of, for example, 448x448 or 896x896 pixels often significantly improves accuracy but slows down DNN refinement. **Using higher image resolution is nearly free lunch and almost always boosts accuracy.**
- DNN over-fitting: Avoid a large gap between the training and test accuracy during DNN refinement (first figure in part 2). This gap can be reduced using dropout rates `rf_dropoutRate` of 0.5 or more, and by increasing the regularizer weight `rf_l2RegWeight`. Using a high dropout rate can be especially helpful if the DNN input image resolution is high.
- Try using deeper DNNs by changing `rf_pretrainedModelFilename` from `ResNet_18.model` to either `ResNet_34.model` or `ResNet_50.model`. The Resnet-50 model is not only deeper, but its output of the penultimate layer is of size 2048 floats (vs. 512 floats of the ResNet-18 and ResNet-34 models). This increased dimension can be especially beneficial when training an SVM classifier.

Part 3 - Custom dataset

In part 1 and 2, we trained and evaluated an image classification model using the provided upper body clothing textures images. We now show how to use a custom user-provided dataset instead.

Using a custom dataset

First, let's have a look at the folder structure for the clothing texture data. Note how all images for the different attributes are in the respective subfolders *dotted*, *leopard*, and *striped* at *DATA_DIR/images/fashionTexture/*. Note also how the image folder name also occurs in the `PARAMETERS.py` file:

```
datasetName = "fashionTexture"
```

Using a custom dataset is as simple as reproducing this folder structure where all images are in subfolders according to their attribute, and to copy these subfolders to a new user-specified directory *DATA_DIR/images/newDataSetName/*. The only code change required is to set the `datasetName` variable to *newDataSetName*. Scripts 1-5 can then be executed in order, and all intermediate files are written to *DATA_DIR/proc/newDataSetName/*. No other code changes are required.

It is important that each image can be assigned to exactly one attribute. For example, it would be wrong to have attributes for 'animal' and for 'leopard', since a 'leopard' image would also belong to 'animal'. Also, it is best to remove images that are ambiguous and hence difficult to annotate.

Image scraping and annotation

Collecting a sufficiently large number of annotated images for training and testing can be difficult. One way to overcome this problem is to scrape images from the Internet.

IMPORTANT

For any images you use, make sure you don't violate the image's copyright and licensing.

To generate a large and diverse dataset, multiple queries should be used. For example, $7 \times 3 = 21$ queries can be synthesized automatically using all combinations of clothing items {blouse, hoodie, pullover, sweater, shirt, t-shirt, vest} and attributes {striped, dotted, leopard}. Downloading the top 50 images per query would then lead to a

maximum of $21 \times 50 = 1050$ images.

Some of the downloaded images are exact or near duplicates (for example, differ just by image resolution or jpg artifacts). These duplicates should be removed so that the training and test split do not contain the same images. Removing duplicate images can be achieved using a hashing-based approach, which works in two steps: (i) first, the hash string is computed for all images; (ii) in a second pass over the images, only those images are kept with a hash string that has not yet been seen. All other images are discarded. We found the `dhash` approach in the Python library `imagehash` and described in this [blog](#) to perform well, with the parameter `hash_size` set to 16. It is OK to incorrectly remove some non-duplicate images, as long as the majority of the real duplicates get removed.

Conclusion

Some key highlights of this example are:

- Code to train, evaluate, and deploy image classification models.
- Demo images provided, but easily adaptable (one line change) to use own image dataset.
- State-of-the-art expert features implemented to train high accuracy models based on Transfer Learning.
- Interactive model development with Azure Machine Learning Workbench and Jupyter Notebook.

References

- [1] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton, *ImageNet Classification with Deep Convolutional Neural Networks*. NIPS 2012.
- [2] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun, *Deep Residual Learning for Image Recognition*. CVPR 2016.

Deep learning for predictive maintenance real-world scenarios

9/24/2018 • 7 minutes to read • [Edit Online](#)

NOTE

This article is deprecated. Support for earlier versions of this service will end incrementally. [View the support timeline](#). Start using the latest version with this [quickstart](#).

Deep learning is one of the most popular trends in machine learning and has applications to many areas, including:

- Driverless cars and robotics.
- Speech and image recognition.
- Financial forecasting.

Also known as deep neural networks (DNN), these methods are inspired by the individual neurons that are within the brain (biological neural networks).

The impact of unscheduled equipment downtime can be detrimental for any business. It's critical to keep field equipment running to maximize utilization and performance and minimize costly, unscheduled downtime. Early identification of issues can help allocate limited maintenance resources in a cost-effective way and enhance quality and supply chain processes.

A predictive maintenance (PM) strategy uses machine learning methods to determine the condition of equipment to preemptively perform maintenance to avoid adverse machine performance. In PM, data is collected over time to monitor the state of the machine and then analyzed to find patterns to predict failures. [Long Short Term Memory \(LSTM\)](#) networks are attractive for this setting since they're designed to learn from sequences of data.

Cortana Intelligence Gallery GitHub repository

The Cortana Intelligence Gallery for the PM tutorial is a public GitHub repository (<https://github.com/Azure/MachineLearningSamples-DeepLearningforPredictiveMaintenance>) where you can report issues and make contributions.

Use case overview

This tutorial uses the example of simulated aircraft engine run-to-failure events to demonstrate the predictive maintenance modeling process. The scenario is described at [Predictive Maintenance](#).

The main assumption in this setting is the engine progressively degraded over its lifetime. The degradation can be detected in engine sensor measurements. PM tries to model the relationship between the changes in these sensor values and the historical failures. The model can then predict when and engine may fail in the future based on the current state of sensor measurements.

This scenario creates an LSTM network to predict the remaining useful life (RUL) of aircraft engines by using historical sensor values. The scenario uses the [Keras](#) library with the [Tensorflow](#) deep learning framework as a compute engine. The scenario trains the LSTM with one set of engines and tests the network on an unseen engine set.

Prerequisites

- An [Azure account](#) (free trials are available).
- Azure Machine Learning Workbench, with a workspace created.
- For model operationalization: Azure Machine Learning Operationalization with a local deployment environment set up, and an [Azure Machine Learning Model Management account](#).

Create a new Workbench project

Create a new project by using this example as a template:

1. Open Machine Learning Workbench.
2. On the **Projects** page, select +, and then select **New Project**.
3. In the **Create New Project** pane, enter the information for your new project.
4. In the **Search Project Templates** search box, type "Predictive Maintenance" and select the **Deep Learning for Predictive Maintenance Scenario** template.
5. Select **Create**.

Prepare the notebook server computation target

To run on your local machine, from the Machine Learning Workbench **File** menu, select either **Open Command Prompt** or **Open PowerShell CLI**. The CLI interface allows you to access your Azure services by using the `az` commands. First, log in to your Azure account with the command:

```
az login
```

This command provides an authentication key to be used with the `https://aka.ms/devicelogin` URL. The CLI waits until the device login operation returns and provides some connection information. Next, if you have a local [Docker](#) installation, prepare the local compute environment with the command:

```
az ml experiment prepare --target docker --run-configuration docker
```

It's preferable to run on a [Data Science Virtual Machine \(DSVM\) for Linux \(Ubuntu\)](#) for memory and disk requirements. After the DSVM is configured, prepare the remote Docker environment with the following two commands:

```
az ml computetarget attach remotedocker --name [Connection_Name] --address [VM_IP_Address] --username [VM_Username] --password [VM_UserPassword]
```

After you're connected to the remote Docker container, prepare the DSVM Docker compute environment with the command:

```
az ml experiment prepare --target [Connection_Name] --run-configuration [Connection_Name]
```

With the Docker compute environment prepared, open the Jupyter notebook server from the Machine Learning Workbench **Notebooks** tab, or start a browser-based server with the command:

```
az ml notebook start
```

The example notebooks are stored in the `Code` directory. The notebooks are set up to run sequentially, starting on the first (`Code\1_data_ingestion.ipynb`) notebook. When you open each notebook, you're prompted to select the compute kernel. Choose the `[Project_Name]_Template [Connection_Name]` kernel to execute on the previously

configured DSVM.

Data description

The template uses three data sets as inputs in the files PM_train.txt, PM_test.txt, and PM_truth.txt.

- **Train data:** The aircraft engine run-to-failure data. The train data (PM_train.txt) consists of multiple, multivariate time series with *cycle* as the time unit. It includes 21 sensor readings for each cycle.
 - Each time series is generated from a different engine of the same type. Each engine starts with different degrees of initial wear and some unique manufacturing variation. This information is unknown to the user.
 - In this simulated data, the engine is assumed to be operating normally at the start of each time series. It starts to degrade at some point during the series of the operating cycles. The degradation progresses and grows in magnitude.
 - When a predefined threshold is reached, the engine is considered unsafe for further operation. The last cycle of each time series is the failure point of that engine.
- **Test data:** The aircraft engine operating data, without failure events recorded. The test data (PM_test.txt) has the same data schema as the training data. The only difference is that the data does not indicate when the failure occurs (the last time period does *not* represent the failure point). It is not known how many more cycles this engine can last before it fails.
- **Truth data:** The information of true remaining cycles for each engine in the testing data. The ground truth data provides the number of remaining working cycles for the engines in the testing data.

Scenario structure

The scenario workflow is divided into the three steps and each step is executed in a Jupyter notebook. Each notebook produces data artifacts that are persisted locally for use in the notebooks.

Task 1: Data ingestion and preparation

The Data Ingestion Jupyter Notebook in Code/1_data_ingestion_and_preparation.ipnyb loads the three input data sets into the Pandas DataFrame format. The notebook then prepares the data for modeling and does some preliminary data visualization. The data sets are stored locally to the compute context for use in the Model Building Jupyter Notebook.

Task 2: Model building and evaluation

The Model Building Jupyter Notebook in Code/2_model_building_and_evaluation.ipnyb reads the train and test data sets from disk and builds an LSTM network for the training data set. The model performance is measured on the test data set. The resulting model is serialized and stored in the local compute context for use in the operationalization task.

Task 3: Operationalization

The Operationalization Jupyter Notebook in Code/3_operationalization.ipnyb uses the stored model to build functions and schema for calling the model on an Azure-hosted web service. The notebook tests the functions and then compresses the assets into the LSTM_o16n.zip file. The file is loaded on to your Azure storage container for deployment.

The LSTM_o16n.zip deployment file contains the following artifacts:

- **webservices_conda.yaml:** Defines the Python packages that are required to run the LSTM model on the deployment target.
- **service_schema.json:** Defines the data schema that's expected by the LSTM model.

- **IstmScore.py**: Defines the functions that the deployment target is running to score new data.
- **modelLSTM.json**: Defines the LSTM architecture. The IstmScore.py functions read the architecture and weights to initialize the model.
- **modelLSTM.h5**: Defines the model weights.
- **test_service.py**: A test script that calls the deployment end point with test data records.
- **PM_test_files.pkl**: The test_service.py script reads historical engine data from the PM_test_files.pkl file and sends the web service enough cycles for the LSTM to return a probability of engine failure.

The notebook tests the functions by using the model definition before it packages the operationalization assets for deployment. Instructions for setting up and testing the web service are included at the end of the Code/3_operationalization.ipynb notebook.

Conclusion

This tutorial provides a simple scenario that uses sensor values to make predictions. More advanced predictive maintenance scenarios like the [Predictive Maintenance Modeling Guide R Notebook](#) can use many data sources, such as historical maintenance records, error logs, and machine features. Additional data sources can require different treatments to use with deep learning.

References

The following references provide examples of other predictive maintenance use cases for various platforms:

- [Predictive Maintenance Solution Template](#)
- [Predictive Maintenance Modeling Guide](#)
- [Predictive Maintenance Modeling Guide using SQL R Services](#)
- [Predictive Maintenance Modeling Guide Python Notebook](#)
- [Predictive Maintenance using PySpark](#)
- [Predictive maintenance real-world scenarios](#)

Next steps

Other example scenarios are available in Machine Learning Workbench that demonstrate additional features of the product.

Data Preparations Python extensions

9/24/2018 • 6 minutes to read • [Edit Online](#)

NOTE

This article is deprecated. Support for earlier versions of this service will end incrementally. [View the support timeline.](#)
Start using the latest version with this [quickstart](#).

As a way of filling in functionality gaps between built-in features, Azure Machine Learning Data Preparations includes extensibility at multiple levels. In this document, we outline the extensibility via Python script.

Custom code steps

Data Preparations has the following custom steps where users can write code:

- Add Column
- Advanced Filter
- Transform Dataflow
- Transform Partition

Code block types

For each of these steps, we support two code block types. First, we support a bare Python Expression that is executed as is. Second, we support a Python Module where we call a particular function with a known signature in the code you supply.

For example, you can add a new column that calculates the log of another column in the following two ways:

Expression

```
math.log(row["Score"])
```

Module

```
def newvalue(row):
    return math.log(row["Score"])
```

The Add Column transform in Module mode expects to find a function called `newvalue` that accepts a row variable and returns the value for the column. This module can include any quantity of Python code with other functions, imports, etc.

The details of each extension point are discussed in the following sections.

Imports

If you use the Expression block type, you can still add `import` statements to your code. They all must be grouped on the top lines of your code.

Correct

```
import math
import numpy
math.log(row["Score"])
```

Error

```
import math
math.log(row["Score"])
import numpy
```

If you use the Module block type, you can follow all the normal Python rules for using the **import** statement.

Default imports

The following imports are always included and usable in your code. You don't need to reimport them.

```
import math
import numbers
import datetime
import re
import pandas as pd
import numpy as np
import scipy as sp
```

Install new packages

To use a package that's not installed by default, you first need to install it into the environments that Data Preparations uses. This installation needs to be done both on your local machine and on any compute targets you want to run on.

To install your packages in a compute target, you have to modify the `conda_dependencies.yml` file located in the `aml_config` folder under the root of your project.

Windows

To find the location on Windows, find the app-specific installation of Python and its scripts directory. The default location is:

```
C:\Users\<user>\AppData\Local\AmlWorkbench\Python\Scripts
```

Then run either of the following commands:

```
conda install <libraryname>
```

or

```
pip install <libraryname>
```

Mac

To find the location on a Mac, find the app-specific installation of Python and its scripts directory. The default location is:

```
/Users/<user>/Library/Caches/AmlWorkbench/Python/bin
```

Then run either of the following commands:

```
./conda install <libraryname>
```

or

```
./pip install <libraryname>
```

Use custom modules

In Transform Dataflow (Script), write the following Python code

```
import sys
sys.path.append(*<absolute path to the directory containing UserModule.py>*)

from UserModule import ExtensionFunction1
df = ExtensionFunction1(df)
```

In Add Column (Script), set Code Block Type = Module, and write the following Python code

```
import sys
sys.path.append(*<absolute path to the directory containing UserModule.py>*)

from UserModule import ExtensionFunction2

def newvalue(row):
    return ExtensionFunction2(row)
```

For different execution contexts (local, Docker, Spark), point absolute path to the right place. You may want to use "os.getcwd() + relativePath" to locate it.

Column data

Column data can be accessed from a row by using dot notation or key-value notation. Column names that contain spaces or special characters can't be accessed by using dot notation. The `row` variable should always be defined in both modes of Python extensions (Module and Expression).

Examples

```
row.ColumnA + row.ColumnB
row["ColumnA"] + row["ColumnB"]
```

Add Column

Purpose

The Add Column extension point lets you write Python to calculate a new column. The code you write has access to the full row. It needs to return a new column value for each row.

How to use

You can add this extension point by using the Add Column (Script) block. It's available on the top-level **Transformations** menu, as well as on the **Column** context menu.

Syntax

Expression

```
math.log(row["Score"])
```

Module

```
def newvalue(row):
    return math.log(row["Score"])
```

Advanced Filter

Purpose

The Advanced Filter extension point lets you write a custom filter. You have access to the entire row, and your code must return True (include the row) or False (exclude the row).

How to use

You can add this extension point by using the Advanced Filter (Script) block. It's available on the top-level **Transformations** menu.

Syntax

Expression

```
row["Score"] > 95
```

Module

```
def includerow(row):
    return row["Score"] > 95
```

Transform Dataflow

Purpose

The Transform Dataflow extension point lets you completely transform the data flow. You have access to a Pandas dataframe that contains all the columns and rows that you're processing. Your code must return a Pandas dataframe with the new data.

NOTE

In Python, all the data to be loaded into memory is in a Pandas dataframe if this extension is used.

In Spark, all the data is collected onto a single worker node. If the data is very large, a worker might run out of memory. Use it carefully.

How to use

You can add this extension point by using the Transform Dataflow (Script) block. It's available on the top-level **Transformations** menu.

Syntax

Expression

```
df['index-column'] = range(1, len(df) + 1)
df = df.reset_index()
```

Module

```
def transform(df):
    df['index-column'] = range(1, len(df) + 1)
    df = df.reset_index()
    return df
```

Transform Partition

Purpose

The Transform Partition extension point lets you transform a partition of the data flow. You have access to a Pandas dataframe that contains all the columns and rows for that partition. Your code must return a Pandas dataframe with the new data.

NOTE

In Python, you might end up with a single partition or multiple partitions, depending on the size of your data. In Spark, you're working with a dataframe that holds the data for a partition on a given worker node. In both cases, you can't assume that you have access to the entire data set.

How to use

You can add this extension point by using the Transform Partition (Script) block. It's available on the top-level **Transformations** menu.

Syntax

Expression

```
df['partition-id'] = index
df['index-column'] = range(1, len(df) + 1)
df = df.reset_index()
```

Module

```
def transform(df, index):
    df['partition-id'] = index
    df['index-column'] = range(1, len(df) + 1)
    df = df.reset_index()
    return df
```

DataPrepError

Error values

In Data Preparations, the concept of error values exists.

It's possible to encounter error values in custom Python code. They are instances of a Python class called `DataPrepError`. This class wraps a Python exception and has a couple of properties. The properties contain information about the error that occurred when the original value was processed, as well as the original value.

DataPrepError class definition

```
class DataPrepError(Exception):
    def __bool__(self):
        return False
```

The creation of a DataPrepError in the Data Preparations Python framework generally looks like this:

```
DataPrepError({
    'message': 'Cannot convert to numeric value',
    'originalValue': value,
    'exceptionMessage': e.args[0],
    '__errorCode__': 'Microsoft.DPrep.ErrorValues.InvalidNumericType'
})
```

How to use

It's possible when Python runs at an extension point to generate DataPrepErrors as return values by using the previous creation method. It's much more likely that DataPrepErrors are encountered when data is processed at an extension point. At this point, the custom Python code needs to handle a DataPrepError as a valid data type.

Syntax

Expression

```
if (isinstance(row["Score"], DataPrepError)):
    row["Score"].originalValue
else:
    row["Score"]
```

```
if (hasattr(row["Score"], "originalValue")):
    row["Score"].originalValue
else:
    row["Score"]
```

Module

```
def newvalue(row):
    if (isinstance(row["Score"], DataPrepError)):
        return row["Score"].originalValue
    else:
        return row["Score"]
```

```
def newvalue(row):
    if (hasattr(row["Score"], "originalValue")):
        return row["Score"].originalValue
    else:
        return row["Score"]
```

Supported data sources for Azure Machine Learning data preparation

10/8/2018 • 3 minutes to read • [Edit Online](#)

NOTE

This article is deprecated. Support for earlier versions of this service will end incrementally. [View the support timeline](#). Start using the latest version with this [quickstart](#).

This article outlines the currently supported data sources for Azure Machine Learning data preparation.

The supported data sources for this release are as follows.

Types

SQL Server

Read from on-prem SQL server or Azure SQL database.

Options

- Server Address
- Trust Server (Even when the certificate on the server is not valid. Use with caution)
- Authentication Type (Windows, Server)
- User Name
- Password
- Database to connect to
- SQL Query

Notes

- Sql-variant columns are not supported
- Time column is converted to datetime by appending time from database to date 1970/1/1
- When executed on Spark cluster, all data related columns (date, datetime, datetime2, datetimeoffset) will evaluate incorrect values for dates before 1583
- Values in decimal columns may lose precision due to conversion to decimal

Directory vs. file

Choose a single file and read it into data preparation. The file type is parsed to determine the default parameters for the file connection shown on the next screen.

Choose a directory or a set of files within a directory (the file picker is multiselect). With either approach, the files are read in as a single data flow and are appended to each other, with headers stripped out if needed.

The supported file types are:

- Delimited (.csv, .tsv, .txt, etc.)
- Fixed width
- Plain text
- JSON file

CSV file

Read a comma-separated-value file from storage.

Options

- Separator
- Comment
- Headers
- Decimal symbol
- File encoding
- Lines to skip

TSV file

Read a tab-separated-value file from storage.

Options

- Comment
- Headers
- File encoding
- Lines to skip

Excel (.xls/.xlsx)

Read an Excel file one sheet at a time by specifying sheet name or number.

Options

- Sheet name or number
- Headers
- Lines to skip

JSON file

Read a JSON file from storage. The file is "flattened" on read.

Options

- None

Parquet

Read a Parquet data set, either a single file or a folder.

Parquet as a format can take various forms in storage. For smaller data sets, a single .parquet file is sometimes used. Various Python libraries support reading or writing to single .parquet files. For the moment, Azure Machine Learning data preparation relies on the PyArrow Python library for reading Parquet during local interactive use. It supports single .parquet files (as long as they were written as such, and not as part of a larger data set), as well as Parquet data sets.

A Parquet data set is a collection of more than one .parquet file, each of which represents a smaller partition of a larger data set. Data sets are usually contained in a folder and are the default parquet output format for platforms such as Spark and Hive.

NOTE

When you're reading Parquet data that's in a folder with multiple .parquet files, it's safest to select the directory for reading, and the **Parquet data set** option. This makes PyArrow read the whole folder instead of the individual files. This ensures support for reading more complicated ways of storing Parquet on disk, such as folder partitioning.

Scale-out execution relies on Spark's Parquet reading capabilities and supports single files as well as folders, similar to local interactive use.

Options

- Parquet data set. This option determines whether Azure Machine Learning data preparation expands a given directory and attempts to read each file individually (the unselected mode), or whether it treats the directory as the whole data set (the selected mode). With the selected mode, PyArrow chooses the best way to interpret the files.

Locations

Local

A local hard drive or a mapped network storage location.

SQL Server

On-prem SQL server, or Azure SQL database.

Azure Blob storage

Azure Blob storage, which requires an Azure subscription.

Connecting to Azure Cosmos DB as a data source

9/24/2018 • 2 minutes to read • [Edit Online](#)

NOTE

This article is deprecated. Support for earlier versions of this service will end incrementally. [View the support timeline](#). Start using the latest version with this [quickstart](#).

This article contains a python sample allows you to connect to Cosmos DB in the Azure Machine Learning Workbench.

Load Azure Cosmos DB data into data preparation

Create a new script-based data flow, and then use the following script to load the data from Azure Cosmos DB.

```
import pydocumentdb
import pydocumentdb.document_client as document_client

import pandas as pd

config = {
    'ENDPOINT': '<Endpoint>',
    'MASTERKEY': '<Key>',
    'DOCUMENTDB_DATABASE': '<DBName>',
    'DOCUMENTDB_COLLECTION': '<collectionname>'
};

# Initialize the Python DocumentDB client.
client = document_client.DocumentClient(config['ENDPOINT'], {'masterKey': config['MASTERKEY']})

# Read databases and take first since id should not be duplicated.
db = next((data for data in client.ReadDatabases() if data['id'] == config['DOCUMENTDB_DATABASE']))

# Read collections and take first since id should not be duplicated.
coll = next((coll for coll in client.ReadCollections(db['_self']) if coll['id'] == config['DOCUMENTDB_COLLECTION']))

docs = client.ReadDocuments(coll['_self'])

df = pd.DataFrame(list(docs))
```

Other data source connections

For other samples, read [Example additional source data connections](#)

Supported inspectors for the Azure Machine Learning data preparation preview

9/24/2018 • 3 minutes to read • [Edit Online](#)

NOTE

This article is deprecated. Support for earlier versions of this service will end incrementally. [View the support timeline.](#)
Start using the latest version with this [quickstart](#).

This document outlines the set of inspectors that are available in this preview.

The halo effect

Some inspectors support the halo effect. This effect uses two different colors to immediately show the change visually from a transform. The gray represents the value prior to the latest transform, and the blue shows the current value. This effect can be enabled and disabled in options.

Graphical filtering

Some of the inspectors support the filtering of data by using the inspector as an editor. Using the inspector as an editor involves selecting graphical elements, and then using the toolbar in the upper-right part of the inspector window to filter in or out the selected values.

Column statistics

For numeric columns, this inspector provides a variety of different stats about the column. Statistics include the following measurements:

- Minimum
- Lower quartile
- Median
- Upper quartile
- Maximum
- Average
- Standard deviation

Options

- None

Histogram

Computes and displays a histogram of a single numeric column. The default number of buckets is calculated using Scott's Rule. However, the rule can be overridden via the options.

This Inspector supports the halo effect.

Options

- Minimum number of buckets (applies even when default bucketing is checked)

- Default number of buckets (Scott's Rule)
- Show halo
- Kernel density plot overlay (Gaussian kernel)
- Use logarithmic scale

Actions

This inspector supports filtering via buckets, which can include single or multi-select buckets. Apply filters as previously described.

Value counts

This inspector presents a frequency table of values for the column that is currently selected. The default display is for the top six values. You can change the limit to any number, however. You can also set the display to count from the bottom instead of the top. This inspector supports the halo effect.

Options

- Number of top values
- Descending
- Include null/error values
- Show halo
- Use logarithmic scale

Actions

This inspector supports filtering via bars, which can include single or multi-select bars. Apply filters as previously described.

Box plot

A box whisker plot of a numeric column.

Options

- Group by column

Scatter plot

A scatter plot for two numeric columns. The data is down-sampled for performance reasons. The sample size can be overridden in the options.

Options

- X-axis column
- Y-axis column
- Sample size
- Group by column

Time series

A line graph with time awareness on the x-axis.

Options

- Date column
- Numeric column
- Sample size

Actions

This inspector supports filtering via a click-and-drag select method to select a range on the graph. After you complete selection, apply filters as previously described.

Map

A map with points that are plotted, assuming that latitude and longitude have been specified. Latitude must be selected first.

Options

- Latitude column
- Longitude column
- Clustering on
- Group by column

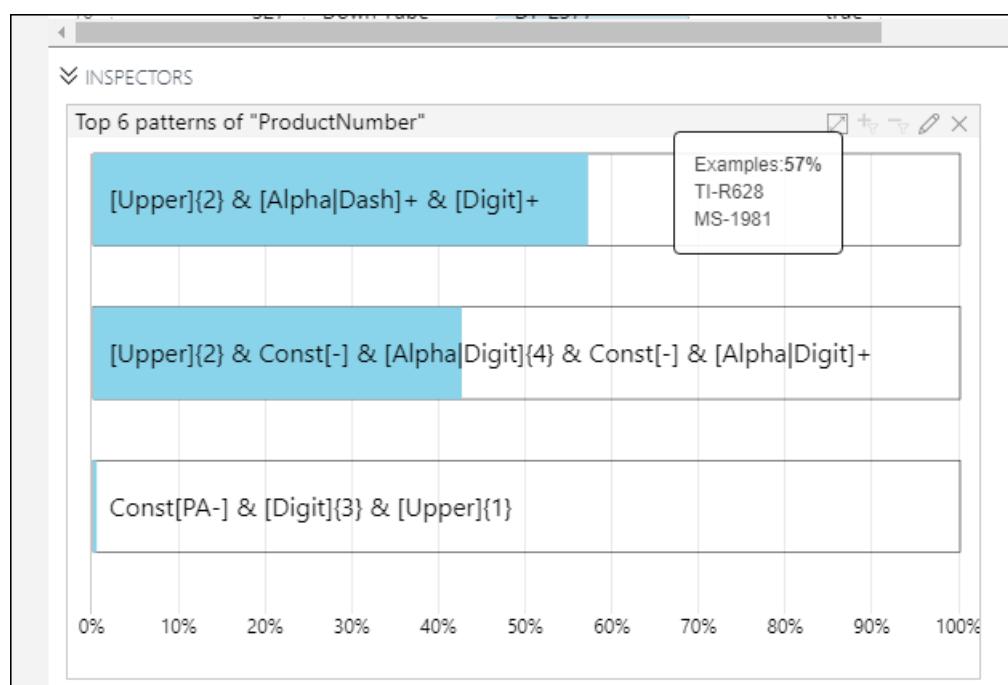
Actions

This inspector supports filtering via point selection on the map. Press the **Ctrl** key, and then click and drag with the mouse to form a square around the points. Then apply filters as previously described.

You can quickly size the map to show only the possible points by pressing the **E** on the left side of the map.

Pattern Frequency

This inspector shows a list of patterns in the selected String column. The patterns are represented using a regular expression like syntax. Hovering on the pattern shows the examples of values represented by that pattern. Along with the patterns, the approximate coverages in terms of percentage is also shown.



Options

- Number of top values
- Descending
- Show halo

Actions

This inspector supports filtering based on displayed patterns. Press the **Ctrl** key, and then select the filled bars in pattern inspector. Then apply filters as previously described. As a result of the user action, an Advanced filter step is

added. You can see and modify the generated Python code by invoking the edit option of the Advanced Filter step.

Supported data exports for this preview

9/24/2018 • 2 minutes to read • [Edit Online](#)

NOTE

This article is deprecated. Support for earlier versions of this service will end incrementally. [View the support timeline](#). Start using the latest version with this [quickstart](#).

It is possible to export to several different formats. You can use these formats to retain the intermediate results of data preparation before you integrate the results into the rest of the Machine Learning workflow.

Types

CSV file

Write a comma-separated-value file to storage.

Options

- Line endings
- Replace nulls with
- Replace errors with
- Separator

Parquet

Write a dataset to storage as Parquet.

Parquet as a format can take various forms in storage. For smaller datasets, a single .parquet file is sometimes used. Various Python libraries support reading and writing to single .parquet files.

Currently, Azure Machine Learning Workbench relies on the PyArrow Python library for writing out Parquet during local interactive use. This means that single-file Parquet is currently the only Parquet output format that's supported during local interactive use.

During scale-out runs (on Spark), Azure Machine Learning Workbench relies on Spark's Parquet reading and writing capabilities. Spark's default output format for Parquet (currently the only one supported) is similar in structure to a Hive dataset. This means that a folder contains many .parquet files that are each a smaller partition of a larger dataset.

Caveats

Parquet as a format is relatively young and has some implementation inconsistencies across different libraries. For instance, Spark places restrictions on which characters are valid in column names when writing out to Parquet.

PyArrow does not do this. The following characters can't be in a column name:

- ,
- ;
- { }
- ()
- \n
- \t
- =

NOTE

- To ensure compatibility with Spark, any time you write data to Parquet, occurrences of these characters in column names are replaced with and underscore (_).
- To ensure consistency across local and scale-out runs, any data written to Parquet, via the app, Python, or Spark, has its column names sanitized to ensure Spark compatibility. To ensure expected column names when writing to Parquet characters in Spark, remove the invalid set from the columns before writing them out.

Locations

Local

Local hard drive or mapped network storage location.

Azure Blob storage

Azure Blob storage requires an Azure subscription.

Supported matrix for this release

9/24/2018 • 2 minutes to read • [Edit Online](#)

NOTE

This article is deprecated. Support for earlier versions of this service will end incrementally. [View the support timeline](#). Start using the latest version with this [quickstart](#).

When your code loads data by using Azure Machine Learning Data Sources or Azure Machine Learning Data Preparations, getting either a Pandas or Spark dataframe, the following combinations of experiment compute environments and data locations are supported:

	LOCAL FILES	AZURE BLOB STORAGE	SQL SERVER DATABASE***
Local Python	Supported	Not supported	Not supported
Docker (Linux VM) Python	Supported in project files only*	Not supported	Not supported
Docker (Linux VM) PySpark	Supported in project files only*	Supported	Supported**
Azure Data Science Virtual Machine Python	Supported in project files only*	Not supported	Not supported
Azure Data Science Virtual Machine PySpark	Supported in project files only*	Not supported	Not supported
Azure HDInsight PySpark	Not supported	Supported	Supported**
Azure HDInsight Python	Not supported	Not supported	Not supported

Azure Data Lake Store is not currently supported for any compute target.

*When local file paths are used, files within the project are copied into the compute environment and then read there. Files outside the project are not copied, and the paths will no longer resolve in the compute environment. Consider using Data Source Substitution so that your code can use a local file when running locally. Then switch to an Azure Storage blob for a different run configuration. You also can use sampling support on data sources to manage runs on large data only in certain run configurations.

**Uses Maven JDBC SQL Server driver 6.2.1. You must ensure that this package (or a compatible one) is included in your spark_dependencies.yml file for the compute environment.

***Supports Azure SQL Database or SQL Server provided the database can be reached from the compute environment.

Sample of filter expressions (Python)

9/24/2018 • 2 minutes to read • [Edit Online](#)

NOTE

This article is deprecated. Support for earlier versions of this service will end incrementally. [View the support timeline](#). Start using the latest version with this [quickstart](#).

Before you read this appendix, read [Python extensibility overview](#).

Filter with equivalence test

Filter in only those rows where the value of (numeric) Col2 is greater than 4.

```
row["Col2"] > 4
```

Filter with multiple columns

Filter in only those rows where Col1 contains the value **Good** and Col2 contains the (numeric) value 1.

```
row["Col1"] == 'Good' and row["Col2"] == 1
```

Test filter against null

Filter in only those rows where Col1 has a null value.

```
pd.isnull(row["Col1"])
```

Sample of custom data flow transforms (Python)

9/24/2018 • 2 minutes to read • [Edit Online](#)

NOTE

This article is deprecated. Support for earlier versions of this service will end incrementally. [View the support timeline](#). Start using the latest version with this [quickstart](#).

The name of the transform in the menu is **Transform Dataflow (Script)**. Before you read this appendix, read [Python extensibility overview](#).

Transform frame

Create a new column dynamically

Creates a column dynamically (**city2**) and reconciles multiple different versions of San Francisco to one from the existing city column.

```
df.loc[(df['city'] == 'San Francisco') | (df['city'] == 'SF') | (df['city'] == 'S.F.') | (df['city'] == 'SAN FRANCISCO'), 'city2'] = 'San Francisco'
```

Add new aggregates

Creates a new frame with the first and last aggregates computed for the score column. These are grouped by the **risk_category** column.

```
df = df.groupby(['risk_category'])['Score'].agg(['first','last'])
```

Winsorize a column

Reformulates the data to meet a formula for reducing the outliers in a column.

```
import scipy.stats as stats
df['Last Order'] = stats.mstats.winsorize(df['Last Order'].values, limits=0.4)
```

Transform data flow

Fill down

Fill down requires two transforms. It assumes data that looks like the following table:

STATE	CITY
Washington	Redmond
	Bellevue
	Issaquah
	Seattle

STATE	CITY
California	Los Angeles
	San Diego
	San Jose
Texas	Dallas
	San Antonio
	Houston

1. Create an "Add Column (Script)" transform using the following code:

```
row['State'] if len(row['State']) > 0 else None
```

2. Create a "Transform Data Flow (Script)" transform that contains the following code:

```
df = df.fillna( method='pad' )
```

The data now looks like the following table:

STATE	NEWSTATE	CITY
Washington	Washington	Redmond
	Washington	Bellevue
	Washington	Issaquah
	Washington	Seattle
California	California	Los Angeles
	California	San Diego
	California	San Jose
Texas	Texas	Dallas
	Texas	San Antonio
	Texas	Houston

Min-max normalization

```
df["NewCol"] = (df["Col1"]-df["Col1"].mean())/df["Col1"].std()
```

Sample of custom source connections (Python)

9/24/2018 • 2 minutes to read • [Edit Online](#)

NOTE

This article is deprecated. Support for earlier versions of this service will end incrementally. [View the support timeline](#). Start using the latest version with this [quickstart](#).

Before you read this appendix, read [Python extensibility overview](#).

Load data from data.world

Prerequisites

Register yourself at data.world

You need an API token from the data.world website.

Install data.world library

Open the Azure Machine Learning Workbench command-line interface by selecting **File > Open command-line interface**.

```
pip install git+git://github.com/datadotworld/data.world-py.git
```

Next, run `dw configure` on the command line, which prompts you for your token. When you enter your token, a `.dw/` directory is created in your home directory and your token is stored there.

```
API token (obtained at: https://data.world/settings/advanced): <enter API token here>
```

Now you should be able to import data.world libraries.

Load data into data preparation

Create a Transform Data Flow (Script) transform. Then use the following script to load the data from data.world.

```
#paths = df['Path'].tolist()

import datadotworld as dw

#Load the dataset.
lds = dw.load_dataset('data-society/the-simpsons-by-the-data')

#Load specific data frame from the dataset.
df = lds.dataframes['simpsons_episodes']
```

Azure Cosmos DB as a data source connection

For an example of Azure Cosmos DB as a data connection, read [Load Azure Cosmos DB as a source data connection](#)

Sample of destination connections (Python)

9/24/2018 • 2 minutes to read • [Edit Online](#)

NOTE

This article is deprecated. Support for earlier versions of this service will end incrementally. [View the support timeline](#). Start using the latest version with this [quickstart](#).

Before you read this appendix, read [Python extensibility overview](#).

Write to Excel

Writing to Excel requires an additional library. Adding new libraries is documented in the extensibility overview.

`openpyxl` is the library that you need to add.

Before you write to Excel, some other changes might be needed. Some of the data types that are used in data preparation are not supported in some destination formats. For example, if "Error" objects exist, they won't serialize correctly to Excel. Thus, before you attempt to write to Excel, you need a "Replace Error Values" transform, which removes errors from any columns.

If all of the previous work is complete, the following line writes the data table to a single sheet in an Excel document. Add a Transform DataFlow (Script) transform. Then enter the following code in an expression section.

On Windows

```
df.to_excel('c:\dev\data\Output\Customer.xlsx', sheet_name='Sheet1')
```

On macOS/OS X

```
df.to_excel('c:/dev/data/Output\Customer.xlsx', sheet_name='Sheet1')
```

Sample of custom column transforms (Python)

9/24/2018 • 2 minutes to read • [Edit Online](#)

NOTE

This article is deprecated. Support for earlier versions of this service will end incrementally. [View the support timeline](#). Start using the latest version with this [quickstart](#).

The name of this transform in the menu is **Add Column (Script)**.

Before you read this appendix, read [Python extensibility overview](#).

Test equivalence and replace values

If the value in Col1 is less than 4, then the new column should have a value of 1. If the value in Col1 is more than 4, the new column has the value 2.

```
1 if row["Col1"] < 4 else 2
```

Current date and time

```
datetime.datetime.now()
```

Typecasting

```
float(row["Col1"]) / float(row["Col2"] - 1)
```

Evaluate for nullness

If Col1 contains a null, then mark the new column as **Bad**. If not, mark it as **Good**.

```
'Bad' if pd.isnull(row["Col1"]) else 'Good'
```

New computed column

```
np.log(row["Col1"])
```

Epoch computation

Number of seconds since the Unix Epoch (assuming Col1 is already a date):

```
row["Col1"] - datetime.datetime.utcnow().timestamp()
```

Hash a column value into a new column

```
import hashlib
hash(row["MyColumnToHashCol1"])
```

Execute Data Sources and Data Preparations packages from Python

9/24/2018 • 2 minutes to read • [Edit Online](#)

NOTE

This article is deprecated. Support for earlier versions of this service will end incrementally. [View the support timeline](#). Start using the latest version with this [quickstart](#).

To use an Azure Machine Learning Data Sources or Azure Machine Learning Data Preparations package within Python:

1. Go to the **Data** tab of your project.
2. Right-click the appropriate source.
3. Choose **Generate Data Access Code File**.

This action creates a short Python script that executes the package and returns a dataframe.

Data Sources

The `azureml.dataprep.datasource` module contains a single function to execute a data source and return a dataframe: `load_datasource(path, secrets=None, spark=None)`.

- `path` is the path to the data source (.dsource file).
- `secrets` is an optional dictionary that maps keys to secrets.
- `spark` is an optional bool that specifies whether to return a Spark dataframe or a Pandas dataframe. By default, Azure Machine Learning Workbench determines which kind of dataframe to return at runtime based on context.

Data Preparations packages

The `azureml.dataprep.package` module contains three functions that execute a data flow from a Data Preparations package.

Execution functions

- `submit(package_path, dataflow_idx=0, secrets=None, spark=None)` submits the specified data flow for execution but does not return a dataframe.
- `run(package_path, dataflow_idx=0, secrets=None, spark=None)` runs the specified data flow and returns the results as a dataframe.
- `run_on_data(user_config, package_path, dataflow_idx=0, secrets=None, spark=None)` runs the specified data flow based on an in-memory data source and returns the results as a dataframe. The `user_config` argument is a dictionary that maps the absolute path of a data source (.dsource file) to an in-memory data source represented as a list of lists.

Common arguments

- `package_path` is the path to the Data Preparations package (.dprep file).
- `dataflow_idx` is the zero-based index of which data flow in the package to execute. If the specified data flow references other data flows or data sources, they are executed as well.

- `secrets` is an optional dictionary that maps keys to secrets.
- `spark` is an optional bool that specifies whether to return a Spark dataframe or a Pandas dataframe. By default, Workbench determines which kind of dataframe to return at runtime based on context.

Azure Machine Learning Experimentation Service configuration files

9/24/2018 • 8 minutes to read • [Edit Online](#)

NOTE

This article is deprecated. Support for earlier versions of this service will end incrementally. [View the support timeline](#). Start using the latest version with this [quickstart](#).

When you run a script in Azure Machine Learning (Azure ML) Workbench, the behavior of the execution is controlled by files in the **aml_config** folder. This folder is under your project folder root. It is important to understand the contents of these files in order to achieve the desired outcome for your execution in an optimal way.

Following are the relevant files under this folder:

- conda_dependencies.yml
- spark_dependencies.yml
- compute target files
 - <compute target name>.compute
- run configuration files
 - <run configuration name>.runconfig

NOTE

You typically have a compute target file and run configuration file for each compute target you create. However, you can create these files independently and have multiple run configuration files pointing to the same compute target.

conda_dependencies.yml

This file is a [conda environment file](#) that specifies the Python runtime version and packages that your code depends on. When Azure ML Workbench executes a script in a Docker container or HDInsight cluster, it creates a [conda environment](#) for your script to run on.

In this file, you specify Python packages that your script needs for execution. Azure ML Experimentation Service creates the conda environment according to your list of dependencies. Packages listed here must be reachable by the execution engine through channels such as:

- [continuum.io](#)
- [PyPI](#)
- a publicly accessible endpoint (URL)
- or a local file path
- others reachable by the execution engine

NOTE

When running on HDInsight cluster, Azure ML Workbench creates a conda environment for your specific run. This allows different users to run on different python environments on the same cluster.

Here is an example of a typical **conda_dependencies.yml** file.

```
name: project_environment
dependencies:
  # Python version
  - python=3.5.2

  # some conda packages
  - scikit-learn
  - cryptography

  # use pip to install some more packages
  - pip:
      # a package in PyPi
      - azure-storage

  # a package hosted in a public URL endpoint
  - https://cntk.ai/PythonWheel/CPU-Only/cntk-2.1-cp35-cp35m-win_amd64.whl

  # a wheel file available locally on disk (this only works if you are executing against local Docker target)
  - C:\temp\my_private_python_pkg.whl
```

Azure ML Workbench uses the same conda environment without rebuilding it as long as the **conda_dependencies.yml** remains the same. It will rebuild your environment if your dependencies change.

NOTE

If you target execution against *local* compute context, **conda_dependencies.yml** file is **not** used. Package dependencies for your local Azure ML Workbench Python environment need to be installed manually.

spark_dependencies.yml

This file specifies the Spark application name when you submit a PySpark script and Spark packages that need to be installed. You can also specify a public Maven repository as well as Spark packages that can be found in those Maven repositories.

Here is an example:

```

configuration:
  # Spark application name
  "spark.app.name": "ClassifyingIris"

repositories:
  # Maven repository hosted in Azure CDN
  - "https://mmlspark.azureedge.net/maven"

  # Maven repository hosted in spark-packages.org
  - "https://spark-packages.org/packages"

packages:
  # MMLSpark package hosted in the Azure CDN Maven
  - group: "com.microsoft.ml.spark"
    artifact: "mmlspark_2.11"
    version: "0.5"

  # spark-sklearn packaged hosted in the spark-packages.org Maven
  - group: "databricks"
    artifact: "spark-sklearn"
    version: "0.2.0"

```

NOTE

Cluster tuning parameters such as worker size and cores should go into "configuration" section in the `spark_dependencies.yml` file

NOTE

If you are executing the script in Python environment, `spark_dependencies.yml` file is ignored. It is used only if you are running against Spark (either on Docker or HDInsight Cluster).

Run configuration

To specify a particular run configuration, you need a `.compute` file and a `.runconfig` file. These are typically generated using a CLI command. You can also clone existing ones, rename them, and edit them.

```

# create a compute target pointing to a VM via SSH
$ az ml computetarget attach remotedocker -n <compute target name> -a <IP address or FQDN of VM> -u
<username> -w <password>

# create a compute context pointing to an HDI cluster head-node via SSH
$ az ml computetarget attach cluster -n <compute target name> -a <IP address or FQDN of HDI cluster> -u
<username> -w <password>

```

This command creates a pair of files based on the compute target specified. Let's say you named your compute target `foo`. This command generates `foo.compute` and `foo.runconfig` in your **aml_config** folder.

NOTE

`local` or `docker` names for the run configuration files are arbitrary. Azure ML Workbench adds these two run configurations when you create a blank project for your convenience. You can rename ".runconfig" files that come with the project template, or create new ones with any name you want.

<compute target name>.compute

`<compute target name>.compute` file specifies connection and configuration information for the compute target.

It is a list of name-value pairs. Following are the supported settings:

type: Type of the compute environment. Supported values are:

- local
- remote
- docker
- remotedocker
- cluster

baseDockerImage: The Docker image used to run the Python/PySpark script. The default value is *microsoft/mmlspark:plus-0.7.91*. We also support one other image: *microsoft/mmlspark:plus-gpu-0.7.91*, which gives you GPU access to the host machine (if GPU is present).

address: The IP address, or FQDN (fully qualified domain name) of the virtual machine, or HDInsight cluster head-node.

username: The SSH username for accessing the virtual machine or the HDInsight head-node.

password: The encrypted password for the SSH connection.

sharedVolumes: Flag to signal that execution engine should use Docker shared volume feature to ship project files back and forth. Having this flag turned on can speed up execution since Docker can access projects directly without the need to copy them. It is best to set *false* if the Docker engine is running on Windows since volume sharing for Docker on Windows can be flaky. Set it to *true* if it is running on macOS or Linux.

nvidiaDocker: This flag, when set to *true*, tells the Azure ML Experimentation Service to use *nvidia-docker* command, as opposed to the regular *docker* command, to launch the Docker image. The *nvidia-docker* engine allows the Docker container to access GPU hardware. The setting is required if you want to run GPU execution in the Docker container. Only Linux host supports *nvidia-docker*. For example, Linux-based DSVM in Azure ships with *nvidia-docker*. *nvidia-docker* as of now is not supported on Windows.

nativeSharedDirectory: This property specifies the base directory (For example: *~/azurerm/share/*) where files can be saved in order to be shared across runs on the same compute target. If this setting is used when running on a Docker container, *sharedVolumes* must be set to true. Otherwise, execution fails.

userManagedEnvironment: This property specifies whether this compute target is managed by the user directly or managed through experimentation service.

pythonLocation: This property specifies the location of the python runtime to be used on the compute target to execute user's program.

<run configuration name>.runconfig

<run configuration name>.runconfig specifies the Azure ML experiment execution behavior. You can configure execution behavior such as tracking run history or what compute target to use along with many others. The names of the run configuration files are used to populate the execution context dropdown in the Azure ML Workbench desktop application.

ArgumentVector: This section specifies the script to be run as part of this execution and the parameters for the script. For example, if you have the following snippet in your ".runconfig" file

```
"ArgumentVector": [  
    - "myscript.py"  
    - 234  
    - "-v"  
]
```

"az ml experiment submit foo.runconfig" automatically runs the command with *myscript.py* file passing in 234 as a parameter and sets the --verbose flag.

Target: This parameter is the name of the *.compute* file that the *runconfig* file references. It generally points the *foo.compute* file but you can edit it to point to a different compute target.

Environment Variables: This section enables users to set environment variables as part of their runs. User can specify environment variables using name-value pairs in the following format:

```
EnvironmentVariables:  
  "EXAMPLE_ENV_VAR1": "Example Value1"  
  "EXAMPLE_ENV_VAR2": "Example Value2"
```

These environment variables can be accessed in user's code. For example, this Python code prints the environment variable named "EXAMPLE_ENV_VAR"

```
print(os.environ.get("EXAMPLE_ENV_VAR1"))
```

Framework: This property specifies if Azure ML Workbench should launch a Spark session to run the script. The default value is *PySpark*. Set it to *Python* if you are not running PySpark code, which can help launching the job quicker with less overhead.

CondaDependenciesFile: This property points to the file that specifies the conda environment dependencies in the *aml_config* folder. If set to *null*, it points to the default **conda_dependencies.yml** file.

SparkDependenciesFile: This property points to the file that specifies the Spark dependencies in the *aml_config* folder. It is set to *null* by default and it points to the default **spark_dependencies.yml** file.

PrepareEnvironment: This property, when set to *true*, tells the Experimentation Service to prepare the conda environment based on the conda dependencies specified as part of your initial run. This property is effective only when you execute against a Docker environment. This setting has no effect if you are running against a *local* environment.

TrackedRun: This flag signals the Experimentation Service whether or not to track the run in Azure ML Workbench run history infrastructure. The default value is *true*.

UseSampling: *UseSampling* specifies whether the active sample datasets for data sources are used for the run. If set to *false*, data sources ingest and use the full data read from the data store. If set to *true*, active samples are used. Users can use the **DataSourceSettings** to specify which specific sample datasets to use if they want to override the active sample.

DataSourceSettings: This configuration section specifies the data source settings. In this section, user specifies which existing data sample for a particular data source is used as part of the run.

The following configuration setting specifies that sample named "MySample" is used for the data source named "MyDataSource"

```
DataSourceSettings:  
  MyDataSource.dsource:  
    Sampling:  
      Sample: MySample
```

DataSourceSubstitutions: Data source substitutions can be used when the user wants to switch from one data source to another without changing their code. For example, users can switch from a sampled-down, local file to the original, larger dataset stored in Azure Blob by changing the data source reference. When a substitution is used, Azure ML Workbench runs your data sources and data preparation packages by referencing the substitute

data source.

The following example replaces the "mylocal.datasource" references in Azure ML data sources and data preparation packages with "myremote.dsource".

```
DataSourceSubstitutions:  
    mylocal.dsource: myremote.dsource
```

Based on the substitution above, the following code sample now reads from "myremote.dsource" instead of "mylocal.dsource" without users changing their code.

```
df = datasource.load_datasource('mylocal.dsource')
```

Next steps

Learn more about [Experimentation Service configuration](#).

Logging API reference

10/30/2018 • 2 minutes to read • [Edit Online](#)

NOTE

This article is deprecated. Support for earlier versions of this service will end incrementally. [View the support timeline](#). Start using the latest version with this [quickstart](#).

Azure ML's logging library allows the program to emit metrics and files that are tracked by the history service for later analysis.

Uploading metrics

```
# import logging API package
from azureml.logging import get_azureml_logger

# initialize a logger object
logger = get_azureml_logger()

# log "scalar" metrics
logger.log("simple integer value", 7)
logger.log("simple float value", 3.141592)
logger.log("simple string value", "this is a string metric")

# log a list of numerical values.
# this automatically creates a chart in the Run History details page
logger.log("chart data points", [1, 3, 5, 10, 6, 4])
```

By default, all metrics are submitted asynchronously so that the submission doesn't impede program execution. This can cause ordering issues when multiple metrics are sent in edge cases. An example of this would be two metrics logged at the same time, but for some reason the user would prefer their exact ordering be preserved. Another case is when the metric must be tracked before running some code that is known to potentially fail fast. In both cases, the solution is to *wait* until the metric is fully logged before proceeding:

```
# blocking call
logger.log("my metric 1", 1).wait()
logger.log("my metric 2", 2).wait()
```

Consuming metrics

The metrics are stored by the history service and tied to the Run that produced them. Both the Run History tab and CLI command below allow you to retrieve them (and artifacts below) after a run has completed.

```
# show the last run
$ az ml history last

# list all past runs
$ az ml history list

# show a particular run
$ az ml history info -r <runid>
```

Artifacts (files)

In addition to metrics, AzureML allows the user to track files as well. By default, all files written into the `outputs` folder relative to the program's working directory (the project folder in the compute context) are uploaded to the history service and tracked for later analysis. The caveat is that the individual file size must be smaller than 512 MB.

```
# Log content as an artifact
logger.upload("artifact/path", "This should be the contents of artifact/path in the service")
```

Consuming artifacts

To print the contents of an artifact that has been tracked, user can use the Run History tab for the given run to **Download** or **Promote** the Artifact, or use the below CLI commands to achieve the same effect.

```
# show all artifacts generated by a run
$ az ml history info -r <runid> -a <artifact/path>

# promote a particular artifact
$ az ml history promote -r <runid> -ap <artifact/prefix> -n <name of asset to create>
```

Next steps

- Walk through the [Classifying iris tutorial, part 2](#) to see logging API in action.
- Review [How to Use Run History and Model Metrics in Azure Machine Learning Workbench](#) to understand deeper how logging APIs can be used in Run History.

Configure the Azure Machine Learning Experimentation Service

9/24/2018 • 2 minutes to read • [Edit Online](#)

NOTE

This article is deprecated. Support for earlier versions of this service will end incrementally. [View the support timeline](#). Start using the latest version with this [quickstart](#).

Overview

Azure Machine Learning Experimentation Service account, workspace, and project are Azure Resources. As such, they can be deployed using Resources Manager templates. Resource Manager templates are JSON files that define the resources you need to deploy for your solution. To understand the concepts associated with deploying and managing your Azure solutions, see [Azure Resource Manager overview](#).

Deploy a template

Deploying a template requires only a couple of steps in the Azure Command Line Interface or in the Azure portal.

Deploy a template from the command line

Using the command-line interface, a single command can deploy a template to an existing resource group. See following for information about creating a template.

```
# Login and validate you are in the right subscription context
az login

# Create a new resource group (you can use an existing one)
az group create --name <resource group name> --location <supported Azure region>
az group deployment create -n testdeploy -g <resource group name> --template-file <template-file.json> --
parameters <parameters.json>
```

Deploy a template from the Azure portal

If you prefer, you can also use Azure portal to create and deploy a template. Do as follows:

1. Navigate to [Azure portal](#).
2. Select **All Services** and search for "templates."
3. Select **Templates**.
4. Click on + **Add** and copy your template information.
5. Select the template created in step #4 and click **Deploy**.

Create a template from an existing Azure resource in the Azure portal

If you already have an Azure Machine experimentation account available, in [Azure portal](#), you can generate a template from that resource.

1. Navigate to an Azure Experimentation Account in [Azure portal](#).
2. Under **settings**, click on **Automation script**.
3. Download the template.

Alternatively, you can manually edit the template files. See following for an example of a template and parameters files.

Template file example

Create a file called "template-file.json" with below content.

```
{
    "contentVersion": "1.0.0.0",
    "$schema": "http://schema.management.azure.com/schemas/2015-01-01/deploymentTemplate.json#",
    "parameters": {
        "accountName": {
            "type": "string",
            "metadata": {
                "description": "Name of the machine learning experimentation team account"
            }
        },
        "location": {
            "type": "string",
            "metadata": {
                "description": "Location of the machine learning experimentation account and other dependent resources."
            }
        },
        "storageAccountSku": {
            "type": "string",
            "defaultValue": "Standard_LRS",
            "metadata": {
                "description": "Sku of the storage account"
            }
        }
    },
    "variables": {
        "mlexpVersion": "2017-05-01-preview",
        "stgResourceId": "[resourceId('Microsoft.Storage/storageAccounts', parameters('accountName'))]"
    },
    "resources": [
        {
            "name": "[parameters('accountName')]",
            "type": "Microsoft.Storage/storageAccounts",
            "location": "[parameters('location')]",
            "apiVersion": "2016-12-01",
            "tags": {
                "mlteamAccount": "[parameters('accountName')]"
            },
            "sku": {
                "name": "[parameters('storageAccountSku')]"
            },
            "kind": "Storage",
            "properties": {
                "encryption": {
                    "services": {
                        "blob": {
                            "enabled": true
                        }
                    },
                    "keySource": "Microsoft.Storage"
                }
            }
        },
        {
            "apiVersion": "[variables('mlexpVersion')]",
            "type": "Microsoft.MachineLearningExperimentation/accounts",
            "name": "[parameters('accountName')]",
            "location": "[parameters('location')]",
            "dependsOn": [
                "[resourceId('Microsoft.Storage/storageAccounts', parameters('accountName'))]"
            ]
        }
    ]
}
```

```
        ],
        "properties": {
            "storageAccount": {
                "storageAccountId": "[variables('stgResourceId')]",
                "accessKey": "[listKeys(resourceId('Microsoft.Storage/storageAccounts', parameters('accountName')), '2016-12-01').keys[0].value]"
            }
        }
    ]
}
```

Parameters

Create a file with below content and save it as <parameters.json>.

There are three values that you can change.

- AccountName: The name of the experimentation account.
- Location: One of the supported Azure regions.
- Storage Account SKU: Azure ML only supports standard storage, not premium. For more information about storage, see [storage introduction](#).

```
{
    "$schema": "https://schema.management.azure.com/schemas/2015-01-01/deploymentParameters.json#",
    "contentVersion": "1.0.0.0",
    "parameters": {
        "accountName": {
            "value": "MyExperimentationAccount"
        },
        "location": {
            "value": "eastus2"
        },
        "storageAccountSku": {
            "value": "Standard_LRS"
        }
    }
}
```

Next steps

- [Create and Install Azure Machine Learning](#)

Azure Machine Learning Model Management Account API reference

9/24/2018 • 20 minutes to read • [Edit Online](#)

NOTE

This article is deprecated. Support for earlier versions of this service will end incrementally. [View the support timeline](#). Start using the latest version with this [quickstart](#).

For information about setting up the deployment environment, see [Model Management account setup](#).

The Azure Machine Learning Model Management Account API implements the following operations:

- Model registration
- Manifest creation
- Docker image creation
- Web service creation

You can use this image to create a web service either locally or on a remote Azure Container Service cluster or another Docker-supported environment of your choice.

Prerequisites

Make sure you have gone through the installation steps in the [Install and create Quickstart](#) document.

The following are required before you proceed:

1. Model Management account provisioning
2. Environment creation for deploying and managing models
3. A Machine Learning model

Azure AD token

When you're using Azure CLI, log in by using `az login`. The CLI uses your Azure Active Directory (Azure AD) token from the .azure file. If you want to use the APIs, you have the following options.

Acquire the Azure AD token manually

You can use `az login` and get the latest token from the .azure file on your home directory.

Acquire the Azure AD token programmatically

```
az ad sp create-for-rbac --scopes /subscriptions/<SubscriptionId>/resourcegroups/<ResourceGroupName> --role Contributor --years <length of time> --name <MyServicePrincipalContributor>
```

After you create the service principal, save the output. Now you can use that to get a token from Azure AD:

```

private static async Task<string> AcquireTokenAsync(string clientId, string password, string authority, string resource)
{
    var creds = new ClientCredential(clientId, password);
    var context = new AuthenticationContext(authority);
    var token = await context.AcquireTokenAsync(resource, creds).ConfigureAwait(false);
    return token.AccessToken;
}

```

The token is put in an authorization header for API calls.

Register a model

The model registration step registers your Machine Learning model with the Azure Model Management account that you created. This registration enables tracking the models and their versions that are assigned at the time of registration. The user provides the name of the model. Subsequent registration of models under the same name generates a new version and ID.

Request

METHOD	REQUEST URI
POST	/api/subscriptions/{subscriptionId}/resourceGroups/{resourceGroupName}/accounts/{accountName}/models

Description

Registers a model.

Parameters

NAME	LOCATED IN	DESCRIPTION	REQUIRED	SCHEMA
subscriptionId	path	Azure subscription ID.	Yes	string
resourceGroupName	path	Name of the resource group in which the Model Management account is located.	Yes	string
accountName	path	Name of the Model Management account.	Yes	string
api-version	query	Version of the Microsoft.MachineLearning resource provider API to use.	Yes	string
Authorization	header	Authorization token. It should be something like "Bearer XXXXXX."	Yes	string
model	body	Payload that is used to register a model.	Yes	Model

Responses

CODE	DESCRIPTION	SCHEMA
200	OK. The model registration succeeded.	Model
default	Error response that describes why the operation failed.	ErrorResponse

Query the list of models in an account

Request

METHOD	REQUEST URI
GET	/api/subscriptions/{subscriptionId}/resourceGroups/{resourceGroupName}/accounts/{accountName}/models

Description

Queries the list of models in an account. You can filter the result list by tag and name. If no filter is passed, the query lists all the models in the account. The returned list is paginated, and the count of items in each page is an optional parameter.

Parameters

NAME	LOCATED IN	DESCRIPTION	REQUIRED	SCHEMA
subscriptionId	path	Azure subscription ID.	Yes	string
resourceGroupName	path	Name of the resource group in which the Model Management account is located.	Yes	string
accountName	path	Name of the Model Management account.	Yes	string
api-version	query	Version of the Microsoft.MachineLearning resource provider API to use.	Yes	string
Authorization	header	Authorization token. It should be something like "Bearer XXXXXX."	Yes	string
name	query	Object name.	No	string
tag	query	Model tag.	No	string
count	query	Number of items to retrieve in a page.	No	string

NAME	LOCATED IN	DESCRIPTION	REQUIRED	SCHEMA
\$skipToken	query	Continuation token to retrieve the next page.	No	string

Responses

CODE	DESCRIPTION	SCHEMA
200	Success.	PaginatedModelList
default	Error response that describes why the operation failed.	ErrorResponse

Get model details

Request

METHOD	REQUEST URI
GET	/api/subscriptions/{subscriptionId}/resourceGroups/{resourceGroupName}/accounts/{accountName}/models/{id}

Description

Gets a model by ID.

Parameters

NAME	LOCATED IN	DESCRIPTION	REQUIRED	SCHEMA
subscriptionId	path	Azure subscription ID.	Yes	string
resourceGroupName	path	Name of the resource group in which the Model Management account is located.	Yes	string
accountName	path	Name of the Model Management account.	Yes	string
id	path	Object ID.	Yes	string
api-version	query	Version of the Microsoft.MachineLearning resource provider API to use.	Yes	string
Authorization	header	Authorization token. It should be something like "Bearer XXXXXX."	Yes	string

Responses

CODE	DESCRIPTION	SCHEMA
200	Success.	Model
default	Error response that describes why the operation failed.	ErrorResponse

Register a manifest with the registered model and all dependencies

Request

METHOD	REQUEST URI
POST	/api/subscriptions/{subscriptionId}/resourceGroups/{resourceGroupName}/accounts/{accountName}/manifests

Description

Registers a manifest with the registered model and all its dependencies.

Parameters

NAME	LOCATED IN	DESCRIPTION	REQUIRED	SCHEMA
subscriptionId	path	Azure subscription ID.	Yes	string
resourceGroupName	path	Name of the resource group in which the Model Management account is located.	Yes	string
accountName	path	Name of the Model Management account.	Yes	string
api-version	query	Version of the Microsoft.MachineLearning resource provider API to use.	Yes	string
Authorization	header	Authorization token. It should be something like "Bearer XXXXXX."	Yes	string
manifestRequest	body	Payload that is used to register a manifest.	Yes	Manifest

Responses

CODE	DESCRIPTION	SCHEMA
200	Manifest registration was successful.	Manifest
default	Error response that describes why the operation failed.	ErrorResponse

Query the list of manifests in an account

Request

METHOD	REQUEST URI
GET	/api/subscriptions/{subscriptionId}/resourceGroups/{resourceGroupName}/accounts/{accountName}/manifests

Description

Queries the list of manifests in an account. You can filter the result list by model ID and manifest name. If no filter is passed, the query lists all the manifests in the account. The returned list is paginated, and the count of items in each page is an optional parameter.

Parameters

NAME	LOCATED IN	DESCRIPTION	REQUIRED	SCHEMA
subscriptionId	path	Azure subscription ID.	Yes	string
resourceGroupName	path	Name of the resource group in which the Model Management account is located.	Yes	string
accountName	path	Name of the Model Management account.	Yes	string
api-version	query	Version of the Microsoft.MachineLearning resource provider API to use.	Yes	string
Authorization	header	Authorization token. It should be something like "Bearer XXXXXX."	Yes	string
modelId	query	Model ID.	No	string
manifestName	query	Manifest name.	No	string
count	query	Number of items to retrieve in a page.	No	string
\$skipToken	query	Continuation token to retrieve the next page.	No	string

Responses

CODE	DESCRIPTION	SCHEMA
200	Success.	PaginatedManifestList

CODE	DESCRIPTION	SCHEMA
default	Error response that describes why the operation failed.	ErrorResponse

Get manifest details

Request

METHOD	REQUEST URI
GET	/api/subscriptions/{subscriptionId}/resourceGroups/{resourceGroupName}/accounts/{accountName}/manifests/{id}

Description

Gets the manifest by ID.

Parameters

NAME	LOCATED IN	DESCRIPTION	REQUIRED	SCHEMA
subscriptionId	path	Azure subscription ID.	Yes	string
resourceGroupName	path	Name of the resource group in which the Model Management account is located.	Yes	string
accountName	path	Name of the Model Management account.	Yes	string
id	path	Object ID.	Yes	string
api-version	query	Version of the Microsoft.MachineLearning resource provider API to use.	Yes	string
Authorization	header	Authorization token. It should be something like "Bearer XXXXXX."	Yes	string

Responses

CODE	DESCRIPTION	SCHEMA
200	Success.	Manifest
default	Error response that describes why the operation failed.	ErrorResponse

Create an image

Request

METHOD	REQUEST URI
POST	/api/subscriptions/{subscriptionId}/resourceGroups/{resourceGroupName}/accounts/{accountName}/images

Description

Creates an image as a Docker image in Azure Container Registry.

Parameters

NAME	LOCATED IN	DESCRIPTION	REQUIRED	SCHEMA
subscriptionId	path	Azure subscription ID.	Yes	string
resourceGroupName	path	Name of the resource group in which the Model Management account is located.	Yes	string
accountName	path	Name of the Model Management account.	Yes	string
api-version	query	Version of the Microsoft.MachineLearning resource provider API to use.	Yes	string
Authorization	header	Authorization token. It should be something like "Bearer XXXXXX."	Yes	string
imageRequest	body	Payload that is used to create an image.	Yes	ImageRequest

Responses

CODE	DESCRIPTION	HEADERS	SCHEMA
202	Async operation location URL. A GET call will show you the status of the image creation task.	Operation-Location	
default	Error response that describes why the operation failed.	ErrorResponse	

Query the list of images in an account

Request

METHOD	REQUEST URI
GET	/api/subscriptions/{subscriptionId}/resourceGroups/{resourceGroupName}/accounts/{accountName}/images

Description

Queries the list of images in an account. You can filter the result list by manifest ID and name. If no filter is passed, the query lists all the images in the account. The returned list is paginated, and the count of items in each page is an optional parameter.

Parameters

NAME	LOCATED IN	DESCRIPTION	REQUIRED	SCHEMA
subscriptionId	path	Azure subscription ID.	Yes	string
resourceGroupName	path	Name of the resource group in which the Model Management account is located.	Yes	string
accountName	path	Name of the Model Management account.	Yes	string
api-version	query	Version of the Microsoft.MachineLearning resource provider API to use.	Yes	string
Authorization	header	Authorization token. It should be something like "Bearer XXXXXX."	Yes	string
manifestId	query	Manifest ID.	No	string
manifestName	query	Manifest name.	No	string
count	query	Number of items to retrieve in a page.	No	string
\$skipToken	query	Continuation token to retrieve the next page.	No	string

Responses

CODE	DESCRIPTION	SCHEMA
200	Success.	PaginatedImageList
default	Error response that describes why the operation failed.	ErrorResponse

Get image details

Request

METHOD	REQUEST URI
GET	/api/subscriptions/{subscriptionId}/resourceGroups/{resourceGroupName}/accounts/{accountName}/images/{id}

Description

Gets an image by ID.

Parameters

NAME	LOCATED IN	DESCRIPTION	REQUIRED	SCHEMA
subscriptionId	path	Azure subscription ID.	Yes	string
resourceGroupName	path	Name of the resource group in which the Model Management account is located.	Yes	string
accountName	path	Name of the Model Management account.	Yes	string
id	path	Image ID.	Yes	string
api-version	query	Version of the Microsoft.MachineLearning resource provider API to use.	Yes	string
Authorization	header	Authorization token. It should be something like "Bearer XXXXXX."	Yes	string

Responses

CODE	DESCRIPTION	SCHEMA
200	Success.	Image
default	Error response that describes why the operation failed.	ErrorResponse

Create a service

Request

METHOD	REQUEST URI
POST	/api/subscriptions/{subscriptionId}/resourceGroups/{resourceGroupName}/accounts/{accountName}/services

Description

Creates a service from an image.

Parameters

NAME	LOCATED IN	DESCRIPTION	REQUIRED	SCHEMA
subscriptionId	path	Azure subscription ID.	Yes	string
resourceGroupName	path	Name of the resource group in which the Model Management account is located.	Yes	string
accountName	path	Name of the Model Management account.	Yes	string
api-version	query	Version of the Microsoft.MachineLearning resource provider API to use.	Yes	string
Authorization	header	Authorization token. It should be something like "Bearer XXXXXX."	Yes	string
serviceRequest	body	Payload that is used to create a service.	Yes	ServiceCreateRequest

Responses

CODE	DESCRIPTION	HEADERS	SCHEMA
202	Async operation location URL. A GET call will show you the status of the service creation task.	Operation-Location	
409	A service with the specified name already exists.		
default	Error response that describes why the operation failed.	ErrorResponse	

Query the list of services in an account

Request

METHOD	REQUEST URI
GET	/api/subscriptions/{subscriptionId}/resourceGroups/{resourceGroupName}/accounts/{accountName}/services

Description

Queries the list of services in an account. You can filter the result list by model name/ID, manifest name/ID, image ID, service name, or Machine Learning compute resource ID. If no filter is passed, the query lists all services in the account. The returned list is paginated, and the count of items in each page is an optional parameter.

Parameters

NAME	LOCATED IN	DESCRIPTION	REQUIRED	SCHEMA
subscriptionId	path	Azure subscription ID.	Yes	string
resourceGroupName	path	Name of the resource group in which the Model Management account is located.	Yes	string
accountName	path	Name of the Model Management account.	Yes	string
api-version	query	Version of the Microsoft.MachineLearning resource provider API to use.	Yes	string
Authorization	header	Authorization token. It should be something like "Bearer XXXXXX."	Yes	string
serviceName	query	Service name.	No	string
modelId	query	Model name.	No	string
modelName	query	Model ID.	No	string
manifestId	query	Manifest ID.	No	string
manifestName	query	Manifest name.	No	string
imageId	query	Image ID.	No	string
computeResourceId	query	Machine Learning compute resource ID.	No	string
count	query	Number of items to retrieve in a page.	No	string
\$skipToken	query	Continuation token to retrieve the next page.	No	string

Responses

CODE	DESCRIPTION	SCHEMA
200	Success.	PaginatedServiceList

CODE	DESCRIPTION	SCHEMA
default	Error response that describes why the operation failed.	ErrorResponse

Get service details

Request

METHOD	REQUEST URI
GET	/api/subscriptions/{subscriptionId}/resourceGroups/{resourceGroupName}/accounts/{accountName}/services/{id}

Description

Gets a service by ID.

Parameters

NAME	LOCATED IN	DESCRIPTION	REQUIRED	SCHEMA
subscriptionId	path	Azure subscription ID.	Yes	string
resourceGroupName	path	Name of the resource group in which the Model Management account is located.	Yes	string
accountName	path	Name of the Model Management account.	Yes	string
id	path	Object ID.	Yes	string
api-version	query	Version of the Microsoft.MachineLearning resource provider API to use.	Yes	string
Authorization	header	Authorization token. It should be something like "Bearer XXXXXX."	Yes	string

Responses

CODE	DESCRIPTION	SCHEMA
200	Success.	ServiceResponse
default	Error response that describes why the operation failed.	ErrorResponse

Update a service

Request

METHOD	REQUEST URI
PUT	/api/subscriptions/{subscriptionId}/resourceGroups/{resourceGroupName}/accounts/{accountName}/services/{id}

Description

Updates an existing service.

Parameters

NAME	LOCATED IN	DESCRIPTION	REQUIRED	SCHEMA
subscriptionId	path	Azure subscription ID.	Yes	string
resourceGroupName	path	Name of the resource group in which the Model Management account is located.	Yes	string
accountName	path	Name of the Model Management account.	Yes	string
id	path	Object ID.	Yes	string
api-version	query	Version of the Microsoft.MachineLearning resource provider API to use.	Yes	string
Authorization	header	Authorization token. It should be something like "Bearer XXXXXX."	Yes	string
serviceUpdateRequest	body	Payload that is used to update an existing service.	Yes	ServiceUpdateRequest

Responses

CODE	DESCRIPTION	HEADERS	SCHEMA
202	Async operation location URL. A GET call will show you the status of the update service task.	Operation-Location	
404	The service with the specified ID does not exist.		
default	Error response that describes why the operation failed.	ErrorResponse	

Delete a service

Request

METHOD	REQUEST URI
DELETE	/api/subscriptions/{subscriptionId}/resourceGroups/{resourceGroupName}/accounts/{accountName}/services/{id}

Description

Deletes a service.

Parameters

NAME	LOCATED IN	DESCRIPTION	REQUIRED	SCHEMA
subscriptionId	path	Azure subscription ID.	Yes	string
resourceGroupName	path	Name of the resource group in which the Model Management account is located.	Yes	string
accountName	path	Name of the Model Management account.	Yes	string
id	path	Object ID.	Yes	string
api-version	query	Version of the Microsoft.MachineLearning resource provider API to use.	Yes	string
Authorization	header	Authorization token. It should be something like "Bearer XXXXXX."	Yes	string

Responses

CODE	DESCRIPTION	SCHEMA
200	Success.	
204	The service with the specified ID does not exist.	
default	Error response that describes why the operation failed.	ErrorResponse

Get service keys

Request

METHOD	REQUEST URI
GET	/api/subscriptions/{subscriptionId}/resourceGroups/{resourceGroupName}/accounts/{accountName}/services/{id}/keys

Description

Gets service keys.

Parameters

NAME	LOCATED IN	DESCRIPTION	REQUIRED	SCHEMA
subscriptionId	path	Azure subscription ID.	Yes	string
resourceGroupName	path	Name of the resource group in which the Model Management account is located.	Yes	string
accountName	path	Name of the Model Management account.	Yes	string
id	path	Service ID.	Yes	string
api-version	query	Version of the Microsoft.MachineLearning resource provider API to use.	Yes	string
Authorization	header	Authorization token. It should be something like "Bearer XXXXXX."	Yes	string

Responses

CODE	DESCRIPTION	SCHEMA
200	Success.	AuthKeys
default	Error response that describes why the operation failed.	ErrorResponse

Regenerate service keys

Request

METHOD	REQUEST URI
POST	/api/subscriptions/{subscriptionId}/resourceGroups/{resourceGroupName}/accounts/{accountName}/services/{id}/regenerateKeys

Description

Regenerates service keys and returns them.

Parameters

NAME	LOCATED IN	DESCRIPTION	REQUIRED	SCHEMA
subscriptionId	path	Azure subscription ID.	Yes	string
resourceGroupName	path	Name of the resource group in which the Model Management account is located.	Yes	string
accountName	path	Name of the Model Management account.	Yes	string
id	path	Service ID.	Yes	string
api-version	query	Version of the Microsoft.MachineLearning resource provider API to use.	Yes	string
Authorization	header	Authorization token. It should be something like "Bearer XXXXXX."	Yes	string
regenerateKeyRequest	body	Payload that is used to update an existing service.	Yes	ServiceRegenerateKey Request

Responses

CODE	DESCRIPTION	SCHEMA
200	Success.	AuthKeys
default	Error response that describes why the operation failed.	ErrorResponse

Query the list of deployments in an account

Request

METHOD	REQUEST URI
GET	/api/subscriptions/{subscriptionId}/resourceGroups/{resourceGroupName}/accounts/{accountName}/deployments

Description

Queries the list of deployments in an account. You can filter the result list by service ID, which will return only the deployments that are created for the particular service. If no filter is passed, the query lists all the deployments in the account.

Parameters

NAME	LOCATED IN	DESCRIPTION	REQUIRED	SCHEMA
subscriptionId	path	Azure subscription ID.	Yes	string
resourceGroupName	path	Name of the resource group in which the Model Management account is located.	Yes	string
accountName	path	Name of the Model Management account.	Yes	string
api-version	query	Version of the Microsoft.MachineLearning resource provider API to use.	Yes	string
Authorization	header	Authorization token. It should be something like "Bearer XXXXXX."	Yes	string
serviceId	query	Service ID.	No	string

Responses

CODE	DESCRIPTION	SCHEMA
200	Success.	DeploymentList
default	Error response that describes why the operation failed.	ErrorResponse

Get deployment details

Request

METHOD	REQUEST URI
GET	/api/subscriptions/{subscriptionId}/resourceGroups/{resourceGroupName}/accounts/{accountName}/deployments/{id}

Description

Gets the deployment by ID.

Parameters

NAME	LOCATED IN	DESCRIPTION	REQUIRED	SCHEMA
subscriptionId	path	Azure subscription ID.	Yes	string
resourceGroupName	path	Name of the resource group in which the Model Management account is located.	Yes	string

NAME	LOCATED IN	DESCRIPTION	REQUIRED	SCHEMA
accountName	path	Name of the Model Management account.	Yes	string
id	path	Deployment ID.	Yes	string
api-version	query	Version of the Microsoft.MachineLearning resource provider API to use.	Yes	string
Authorization	header	Authorization token. It should be something like "Bearer XXXXXX."	Yes	string

Responses

CODE	DESCRIPTION	SCHEMA
200	Success.	Deployment
default	Error response that describes why the operation failed.	ErrorResponse

Get operation details

Request

METHOD	REQUEST URI
GET	/api/subscriptions/{subscriptionId}/resourceGroups/{resourceGroupName}/accounts/{accountName}/operations/{id}

Description

Gets the async operation status by operation ID.

Parameters

NAME	LOCATED IN	DESCRIPTION	REQUIRED	SCHEMA
subscriptionId	path	Azure subscription ID.	Yes	string
resourceGroupName	path	Name of the resource group in which the Model Management account is located.	Yes	string
accountName	path	Name of the Model Management account.	Yes	string
id	path	Operation ID.	Yes	string

NAME	LOCATED IN	DESCRIPTION	REQUIRED	SCHEMA
api-version	query	Version of the Microsoft.MachineLearning resource provider API to use.	Yes	string
Authorization	header	Authorization token. It should be something like "Bearer XXXXXX."	Yes	string

Responses

CODE	DESCRIPTION	SCHEMA
200	Success.	OperationStatus
default	Error response that describes why the operation failed.	ErrorResponse

Definitions

Asset

The asset object that will be needed during Docker image creation.

NAME	DESCRIPTION	SCHEMA
id <i>optional</i>	Asset ID.	string
 mimeType <i>optional</i>	MIME type of model content. For more information about MIME type, see the list of IANA media types .	string
unpack <i>optional</i>	Indicates where we need to unpack the content during Docker image creation.	boolean
url <i>optional</i>	Asset location URL.	string

AsyncOperationState

The async operation state.

Type: enum (NotStarted, Running, Cancelled, Succeeded, Failed)

AsyncOperationStatus

The operation status.

NAME	DESCRIPTION	SCHEMA
createdTime <i>optional</i> <i>read-only</i>	Async operation creation time (UTC).	string (date-time)

NAME	DESCRIPTION	SCHEMA
endTime <i>optional read-only</i>	Async operation end time (UTC).	string (date-time)
error <i>optional</i>		ErrorResponse
id <i>optional</i>	Async operation ID.	string
operationType <i>optional</i>	Async operation type.	enum (Image, Service)
resourceLocation <i>optional</i>	Resource created or updated by the async operation.	string
state <i>optional</i>		AsyncOperationState

AuthKeys

The authentication keys for a service.

NAME	DESCRIPTION	SCHEMA
primaryKey <i>optional</i>	Primary key.	string
secondaryKey <i>optional</i>	Secondary key.	string

AutoScaler

Settings for the autoscaler.

NAME	DESCRIPTION	SCHEMA
autoscaleEnabled <i>optional</i>	Enable or disable the autoscaler.	boolean
maxReplicas <i>optional</i>	Maximum number of pod replicas to scale up to. Minimum value: <input type="text" value="1"/>	integer
minReplicas <i>optional</i>	Minimum number of pod replicas to scale down to. Minimum value: <input type="text" value="0"/>	integer
refreshPeriodInSeconds <i>optional</i>	Refresh time for autoscaling trigger. Minimum value: <input type="text" value="1"/>	integer

NAME	DESCRIPTION	SCHEMA
targetUtilization <i>optional</i>	Utilization percentage that triggers autoscaling. Minimum value: <input type="text" value="0"/> Maximum value: <input type="text" value="100"/>	integer

ComputeResource

The Machine Learning compute resource.

NAME	DESCRIPTION	SCHEMA
id <i>optional</i>	Resource ID.	string
type <i>optional</i>	Type of resource.	enum (Cluster)

ContainerResourceReservation

Configuration to reserve resources for a container in the cluster.

NAME	DESCRIPTION	SCHEMA
cpu <i>optional</i>	Specifies CPU reservation. Format for Kubernetes: see Meaning of CPU .	string
memory <i>optional</i>	Specifies memory reservation. Format for Kubernetes: see Meaning of memory .	string

Deployment

An instance of an Azure Machine Learning deployment.

NAME	DESCRIPTION	SCHEMA
createdAt <i>optional</i> <i>read-only</i>	Deployment creation time (UTC).	string (date-time)
expiredAt <i>optional</i> <i>read-only</i>	Deployment expired time (UTC).	string (date-time)
id <i>optional</i>	Deployment ID.	string
imageId <i>optional</i>	Image ID associated with this deployment.	string
serviceName <i>optional</i>	Service name.	string
status <i>optional</i>	Current deployment status.	string

DeploymentList

An array of deployment objects.

Type: <[Deployment](#)> array

ErrorDetail

Model Management service error detail.

NAME	DESCRIPTION	SCHEMA
code <i>required</i>	Error code.	string
message <i>required</i>	Error message.	string

ErrorResponse

A Model Management service error object.

NAME	DESCRIPTION	SCHEMA
code <i>required</i>	Error code.	string
details <i>optional</i>	Array of error detail objects.	< ErrorDetail > array
message <i>required</i>	Error message.	string
statusCode <i>optional</i>	HTTP status code.	integer

Image

The Azure Machine Learning image.

NAME	DESCRIPTION	SCHEMA
computeResourceId <i>optional</i>	ID of the environment created in the Machine Learning compute resource.	string
createdTime <i>optional</i>	Image creation time (UTC).	string (date-time)
creationState <i>optional</i>		AsyncOperationState
description <i>optional</i>	Image description text.	string
error <i>optional</i>		ErrorResponse
id <i>optional</i>	Image ID.	string

NAME	DESCRIPTION	SCHEMA
imageBuildLogUri <i>optional</i>	URI of the uploaded logs from the image build.	string
imageLocation <i>optional</i>	Azure Container Registry location string for the created image.	string
imageType <i>optional</i>		ImageType
manifest <i>optional</i>		Manifest
name <i>optional</i>	Image name.	string
version <i>optional</i>	Image version set by the Model Management service.	integer

ImageRequest

A request to create an Azure Machine Learning image.

NAME	DESCRIPTION	SCHEMA
computeResourceId <i>required</i>	ID of the environment created in the Machine Learning compute resource.	string
description <i>optional</i>	Image description text.	string
imageType <i>required</i>		ImageType
manifestId <i>required</i>	ID of the manifest from which the image will be created.	string
name <i>required</i>	Image name.	string

ImageType

Specifies the type of the image.

Type: enum (Docker)

Manifest

The Azure Machine Learning manifest.

NAME	DESCRIPTION	SCHEMA
assets <i>required</i>	List of assets.	< Asset > array

NAME	DESCRIPTION	SCHEMA
createdTime <i>optional read-only</i>	Manifest creation time (UTC).	string (date-time)
description <i>optional</i>	Manifest description text.	string
driverProgram <i>required</i>	Driver program of the manifest.	string
id <i>optional</i>	Manifest ID.	string
modelIds <i>optional</i>	List of model IDs of the registered models. The request will fail if any of the included models are not registered.	array
modelType <i>optional</i>	Specifies that the models are already registered with the Model Management service.	enum (Registered)
name <i>required</i>	Manifest name.	string
targetRuntime <i>required</i>		TargetRuntime
version <i>optional read-only</i>	Manifest version assigned by the Model Management service.	integer
webserviceType <i>optional</i>	Specifies the desired type of web service that will be created from the manifest.	enum (Realtime)

Model

An instance of an Azure Machine Learning model.

NAME	DESCRIPTION	SCHEMA
createdAt <i>optional read-only</i>	Model creation time (UTC).	string (date-time)
description <i>optional</i>	Model description text.	string
id <i>optional read-only</i>	Model ID.	string
 mimeType <i>required</i>	MIME type of the model content. For more information about MIME type, see the list of IANA media types .	string

NAME	DESCRIPTION	SCHEMA
name <i>required</i>	Model name.	string
tags <i>optional</i>	Model tag list.	array
unpack <i>optional</i>	Indicates whether we need to unpack the model during Docker image creation.	boolean
url <i>required</i>	URL of the model. Usually we put a shared access signature URL here.	string
version <i>optional</i> <i>read-only</i>	Model version assigned by the Model Management service.	integer

ModelDataCollection

The model data collection information.

NAME	DESCRIPTION	SCHEMA
eventHubEnabled <i>optional</i>	Enable an event hub for a service.	boolean
storageEnabled <i>optional</i>	Enable storage for a service.	boolean

PaginatedImageList

A paginated list of images.

NAME	DESCRIPTION	SCHEMA
nextLink <i>optional</i>	Continuation link (absolute URI) to the next page of results in the list.	string
value <i>optional</i>	Array of model objects.	< Image > array

PaginatedManifestList

A paginated list of manifests.

NAME	DESCRIPTION	SCHEMA
nextLink <i>optional</i>	Continuation link (absolute URI) to the next page of results in the list.	string
value <i>optional</i>	Array of manifest objects.	< Manifest > array

PaginatedModelList

A paginated list of models.

NAME	DESCRIPTION	SCHEMA
nextLink <i>optional</i>	Continuation link (absolute URI) to the next page of results in the list.	string
value <i>optional</i>	Array of model objects.	<Model> array

PaginatedServiceList

A paginated list of services.

NAME	DESCRIPTION	SCHEMA
nextLink <i>optional</i>	Continuation link (absolute URI) to the next page of results in the list.	string
value <i>optional</i>	Array of service objects.	<ServiceResponse> array

ServiceCreateRequest

A request to create a service.

NAME	DESCRIPTION	SCHEMA
appInsightsEnabled <i>optional</i>	Enable application insights for a service.	boolean
autoScaler <i>optional</i>		AutoScaler
computeResource <i>required</i>		ComputeResource
containerResourceReservation <i>optional</i>		ContainerResourceReservation
dataCollection <i>optional</i>		ModelDataCollection
imageId <i>required</i>	Image to create the service.	string
maxConcurrentRequestsPerContainer <i>optional</i>	Maximum number of concurrent requests. Minimum value: <input type="text" value="1"/>	integer
name <i>required</i>	Service name.	string
numReplicas <i>optional</i>	Number of pod replicas running at any time. Cannot be specified if Autoscaler is enabled. Minimum value: <input type="text" value="0"/>	integer

ServiceRegenerateKeyRequest

A request to regenerate a key for a service.

NAME	DESCRIPTION	SCHEMA
keyType <i>optional</i>	Specifies which key to regenerate.	enum (Primary, Secondary)

ServiceResponse

The detailed status of the service.

NAME	DESCRIPTION	SCHEMA
createdAt <i>optional</i>	Service creation time (UTC).	string (date-time)
ID <i>optional</i>	Service ID.	string
image <i>optional</i>		Image
manifest <i>optional</i>		Manifest
models <i>optional</i>	List of models.	<Model> array
name <i>optional</i>	Service name.	string
scoringUri <i>optional</i>	URI for scoring the service.	string
state <i>optional</i>		AsyncOperationState
updatedAt <i>optional</i>	Last update time (UTC).	string (date-time)
appInsightsEnabled <i>optional</i>	Enable application insights for a service.	boolean
autoScaler <i>optional</i>		AutoScaler
computeResource <i>required</i>		ComputeResource
containerResourceReservation <i>optional</i>		ContainerResourceReservation
dataCollection <i>optional</i>		ModelDataCollection

NAME	DESCRIPTION	SCHEMA
maxConcurrentRequestsPerContainer <i>optional</i>	Maximum number of concurrent requests. Minimum value: <input type="text" value="1"/>	integer
numReplicas <i>optional</i>	Number of pod replicas running at any time. Cannot be specified if Autoscaler is enabled. Minimum value: <input type="text" value="0"/>	integer
error <i>optional</i>		ErrorResponse

ServiceUpdateRequest

A request to update a service.

NAME	DESCRIPTION	SCHEMA
appInsightsEnabled <i>optional</i>	Enable application insights for a service.	boolean
autoScaler <i>optional</i>		AutoScaler
containerResourceReservation <i>optional</i>		ContainerResourceReservation
dataCollection <i>optional</i>		ModelDataCollection
imageId <i>optional</i>	Image to create the service.	string
maxConcurrentRequestsPerContainer <i>optional</i>	Maximum number of concurrent requests. Minimum value: <input type="text" value="1"/>	integer
numReplicas <i>optional</i>	Number of pod replicas running at any time. Cannot be specified if Autoscaler is enabled. Minimum value: <input type="text" value="0"/>	integer

TargetRuntime

The type of the target runtime.

NAME	DESCRIPTION	SCHEMA
properties <i>required</i>		<string, string> map
runtimeType <i>required</i>	Specifies the runtime.	enum (SparkPython, Python)

Model management command-line interface reference

10/15/2018 • 10 minutes to read • [Edit Online](#)

NOTE

This article is deprecated. Support for earlier versions of this service will end incrementally. [View the support timeline](#). Start using the latest version with this [quickstart](#).

Base CLI concepts:

```
account : Manage model management accounts.  
env     : Manage compute environments.  
image   : Manage operationalization images.  
manifest: Manage operationalization manifests.  
model   : Manage operationalization models.  
service : Manage operationalized services.
```

Account commands

A model management account is required to use the services, which allow you to deploy and manage models. Use

```
az ml account modelmanagement -h
```

```
create: Create a Model Management Account.  
delete: Delete a specified Model Management Account.  
list  : Gets the Model Management Accounts in the current subscription.  
set   : Set the active Model Management Account.  
show  : Show a Model Management Account.  
update: Update an existing Model Management Account.
```

Create Model Management Account

Create a model management account for billing using the following command:

```
az ml account modelmanagement create --location [Azure region e.g. eastus2] --name [new account name] --resource-group [resource group name to store the account in]
```

Local Arguments:

```
--location -l      [Required]: Resource location.  
--name -n         [Required]: Name of the model management account.  
--resource-group -g [Required]: Resource group to create the model management account in.  
--description -d           : Description of the model management account.  
--sku-instances       : Number of instances of the selected SKU. Must be between 1 and  
                         16 inclusive. Default: 1.  
--sku-name            : SKU name. Valid names are S1|S2|S3|DevTest. Default: S1.  
--tags -t             : Tags for the model management account. Default: {}.  
-v                  : Verbosity flag.
```

Environment commands

```

cluster      : Switch the current execution context to 'cluster'.
delete       : Delete an MLCRP-provisioned resource.
get-credentials: List the keys for an environment.
list         : List all environments in the current subscription.
local        : Switch the current execution context to 'local'.
set          : Set the active MLC environment.
setup        : Sets up an MLC environment.
show         : Show an MLC resource; if resource_group or cluster_name are not provided, shows
               the active MLC env.

```

Set up the Deployment Environment

The setup command requires you to have Contributor access to the subscription. If you don't have that, you at least need Contributor access to the resource group that you are deploying into. To do the latter, you need to specify the resource group name as part of the setup command using `-g` the flag.

There are two options for deployment: *local* and *cluster*. Setting the `--cluster` (or `-c`) flag enables cluster deployment, which provisions an ACS cluster. The basic setup syntax is as follows:

```
az ml env setup [-c] --location [location of environment resources] --name[name of environment]
```

This command initializes your Azure machine learning environment with a storage account, ACR registry, and App Insights service created in your subscription. By default, the environment is initialized for local deployments only (no ACS) if no flag is specified. If you need to scale service, specify the `--cluster` (or `-c`) flag to create an ACS cluster.

Command details:

```

--location -l      [Required]: Location for environment resources; an Azure region, e.g. eastus2.
--name -n          [Required]: Name of environment to provision.
--acr -r           : ARM ID of ACR to associate with this environment.
--agent-count -z   : Number of agents to provision in the ACS cluster. Default: 3.
--cert-cname       : CNAME of certificate.
--cert-pem         : Path to .pem file with certificate bytes.
--cluster -c        : Flag to provision ACS cluster. Off by default; specify this to force an ACS
cluster deployment.
--key-pem          : Path to .pem file with certificate key.
--master-count -m  : Number of master nodes to provision in the ACS cluster. Acceptable values: 1,
3, 5. Default: 1.
--resource-group -g : Resource group in which to create compute resource. Is created if it does not
exist.
                                         If not provided, resource group is created with 'rg' appended to 'name.'.
--service-principal-app-id -a : App ID of service principal to use for configuring ML compute.
--service-principal-password -p: Password associated with service principal.
--storage -s         : ARM ID of storage account to associate with this environment.
--yes -y             : Flag to answer 'yes' to any prompts. Command fails if user is not logged in.

```

Global Arguments

```

--debug            : Increase logging verbosity to show all debug logs.
--help -h          : Show this help message and exit.
--output -o        : Output format. Allowed values: json, jsonc, table, tsv. Default: json.
--query            : JMESPath query string. See http://jmespath.org/ for more information and
examples.
--verbose          : Increase logging verbosity. Use --debug for full debug logs.

```

Model commands

```
list  
register  
show
```

Register a model

Command to register the model.

```
az ml model register --model [path to model file] --name [model name]
```

Command details:

```
--model -m [Required]: Model to register.  
--name -n [Required]: Name of model to register.  
--description -d : Description of the model.  
--tag -t : Tags for the model. Multiple tags can be specified with additional -t arguments.  
-v : Verbosity flag.
```

Global Arguments

```
--debug : Increase logging verbosity to show all debug logs.  
--help -h : Show this help message and exit.  
--output -o : Output format. Allowed values: json, jsonc, table, tsv. Default: json.  
--query : JMESPath query string. See http://jmespath.org/ for more information and examples.  
--verbose : Increase logging verbosity. Use --debug for full debug logs.
```

Manifest commands

```
create: Create an Operationalization Manifest. This command has two different sets of required arguments, depending on if you want to use previously registered model/s.
```

```
list  
show
```

Create manifest

The following command creates a manifest file for the model.

```
az ml manifest create --manifest-name [your new manifest name] -f [path to score file] -r [runtime for the image, e.g. spark-py]
```

Command details:

```
--manifest-name -n [Required]: Name of the manifest to create.  
-f [Required]: The score file to be deployed.  
-r [Required]: Runtime of the web service. Valid runtimes are spark-py|python.  
--conda-file -c : Path to Conda Environment file.  
--dependency -d : Files and directories required by the service. Multiple dependencies can be specified with additional -d arguments.  
--manifest-description : Description of the manifest.  
--schema-file -s : Schema file to add to the manifest.  
-p : A pip requirements.txt file needed by the score file.  
-v : Verbosity flag.
```

Registered Model Arguments

```
--model-id -i : [Required] Id of previously registered model to add to manifest.  
Multiple models can be specified with additional -i arguments.
```

Unregistered Model Arguments

```
--model-file -m : [Required] Model file to register. If used, must be combined with  
model name.
```

Global Arguments

```
--debug : Increase logging verbosity to show all debug logs.  
--help -h : Show this help message and exit.  
--output -o : Output format. Allowed values: json, jsonc, table, tsv.  
               Default: json.  
--query : JMESPath query string. See http://jmespath.org/ for more  
               information and examples.  
--verbose : Increase logging verbosity. Use --debug for full debug logs.
```

Image commands

```
create: Creates a docker image with the model and its dependencies. This command has two different sets of  
required arguments, depending on if you want to use a previously created manifest.  
list  
show  
usage
```

Create image

You can create an image with the option of having created its manifest before.

```
az ml image create -n [image name] --manifest-id [the manifest ID]
```

Or you can create the manifest and image with a single command.

```
az ml image create -n [image name] --model-file [model file or folder path] -f [score file, e.g. the score.py  
file] -r [the runtime eg.g. spark-py which is the Docker container image base]
```

Command details:

```
--image-name -n [Required]: The name of the image being created.  
--image-description : Description of the image.  
--image-type : The image type to create. Defaults to "Docker".  
-v : Verbosity flag.
```

Registered Manifest Arguments

```
--manifest-id : [Required] Id of previously registered manifest to use in image creation.
```

Unregistered Manifest Arguments

```
--conda-file -c : Path to Conda Environment file.
--dependency -d : Files and directories required by the service. Multiple dependencies can
                  be specified with additional -d arguments.
--model-file -m : [Required] Model file to register.
--schema-file -s : Schema file to add to the manifest.
-f : [Required] The score file to be deployed.
-p : A pip requirements.txt file needed by the score file.
-r : [Required] Runtime of the web service. Valid runtimes are python|spark-py.
```

Service commands

The following commands are supported for Service. To see the parameters for each command, use the -h option. For example, use `az ml service create realtime -h` to see create command details.

```
create
delete
keys
list
logs
run
show
update
usage
```

Create a service

To create a service with a previously created image, use the following command:

```
az ml service create realtime --image-id [image to deploy] -n [service name]
```

To create a service, manifest, and image with a single command, use the following command:

```
az ml service create realtime --model-file [path to model file(s)] -f [path to model scoring file, e.g.
score.py] -n [service name] -r [run time included in the image, e.g. spark-py]
```

Commands details:

```
-n : [Required] Webservice name.
--autoscale-enabled : Enable automatic scaling of service replicas based on request demand.
                      Allowed values: true, false. False if omitted. Default: false.
--autoscale-max-replicas : If autoscale is enabled - sets the maximum number of replicas.
--autoscale-min-replicas : If autoscale is enabled - sets the minimum number of replicas.
--autoscale-refresh-period-seconds: If autoscale is enabled - the interval of evaluating scaling demand.
--autoscale-target-utilization : If autoscale is enabled - target utilization of replicas time.
--collect-model-data : Enable model data collection. Allowed values: true, false. False if
omitted. Default: false.
--cpu : Reserved number of CPU cores per service replica (can be fraction).
--enable-app-insights -l : Enable app insights. Allowed values: true, false. False if omitted.
Default: false.
--memory : Reserved amount of memory per service replica, in M or G. (ex. 1G, 300M).
--replica-max-concurrent-requests : Maximum number of concurrent requests that can be routed to a service
replica.
-v : Verbosity flag.
-z : Number of replicas for a Kubernetes service. Default: 1.
```

Registered Image Arguments

```
--image-id : [Required] Image to deploy to the service.
```

Unregistered Image Arguments

```
--conda-file -c : Path to Conda Environment file.  
--image-type : The image type to create. Defaults to "Docker".  
--model-file -m : [Required] The model to be deployed.  
-d : Files and directories required by the service. Multiple dependencies can  
be specified  
      with additional -d arguments.  
-f : [Required] The score file to be deployed.  
-p : A pip requirements.txt file of package needed by the score file.  
-r : [Required] Runtime of the web service. Valid runtimes are python|spark-py.  
-s : Input and output schema of the web service.
```

Global Arguments

```
--debug : Increase logging verbosity to show all debug logs.  
--help -h : Show this help message and exit.  
--output -o : Output format. Allowed values: json, jsonc, table, tsv. Default: json.  
--query examples. : JMESPath query string. See http://jmespath.org/ for more information and  
examples.  
--verbose : Increase logging verbosity. Use --debug for full debug logs.
```

Note on the `-d` flag for attaching dependencies: If you pass the name of a directory that is not already bundled (zip, tar, etc.), that directory automatically gets tar'ed and is passed along, then automatically unbundled on the other end.

If you pass in a directory that is already bundled, the directory is treated as a file and passed along as is. It is unbundled automatically; you are expected to handle that in your code.

Get service details

Get service details including URL, usage (including sample data if a schema was created).

```
az ml service show realtime --name [service name]
```

Command details:

```
--id -i : The service id to show.  
--name -n : Webservice name.  
-v : Verbosity flag.
```

Global Arguments

```
--debug : Increase logging verbosity to show all debug logs.  
--help -h : Show this help message and exit.  
--output -o: Output format. Allowed values: json, jsonc, table, tsv. Default: json.  
--query : JMESPath query string. See http://jmespath.org/ for more information and examples.  
--verbose : Increase logging verbosity. Use --debug for full debug logs.
```

Run the service

```
az ml service run realtime -n [service name] -d [input_data]
```

Command details:

```
--id -i : The service id to show.  
--name -n : Webservice name.  
-v : Verbosity flag.
```

Global Arguments

```
--debug    : Increase logging verbosity to show all debug logs.  
--help -h  : Show this help message and exit.  
--output -o: Output format. Allowed values: json, jsonc, table, tsv. Default: json.  
--query   : JMESPath query string. See http://jmespath.org/ for more information and examples.  
--verbose  : Increase logging verbosity. Use --debug for full debug logs.
```

Command

```
az ml service run realtime
```

Arguments --id -i : [Required] The service id to score against. -d : The data to use for calling the web service. -v : Verbosity flag.

Global Arguments

```
--debug    : Increase logging verbosity to show all debug logs.  
--help -h  : Show this help message and exit.  
--output -o: Output format. Allowed values: json, jsonc, table, tsv. Default: json.  
--query   : JMESPath query string. See http://jmespath.org/ for more information and examples.  
--verbose  : Increase logging verbosity. Use --debug for full debug logs.
```

Azure Machine Learning Model Data Collection API reference

9/24/2018 • 2 minutes to read • [Edit Online](#)

NOTE

This article is deprecated. Support for earlier versions of this service will end incrementally. [View the support timeline](#). Start using the latest version with this [quickstart](#).

Model data collection allows you to archive model inputs and predictions from a machine learning web service. See the [model data collection how-to guide](#) to understand how to install `azureml.datacollector` on your Windows and Linux machine.

In this API reference guide, we use a step-by-step approach on how to collect model inputs and predictions from a machine learning web service.

Enable model data collection in Azure ML Workbench environment

Look for `conda_dependencies.yml` file in your project under the `aml_config` folder and have your `conda_dependencies.yml` file include the `azureml.datacollector` module under the pip section as shown below. Note that this is only a subsection of a full `conda_dependencies.yml` file:

```
dependencies:  
  - python=3.5.2  
  - pip:  
    - azureml.datacollector==0.1.0a13
```

NOTE

Currently, you can use the data collector module in Azure ML Workbench by running in docker mode. Local mode may not work for all environments.

Enable model data collection in the scoring file

In the scoring file that is being used for operationalization, import the data collector module and `ModelDataCollector` class:

```
from azureml.datacollector import ModelDataCollector
```

Model data collector instantiation

Instantiate a new instance of a `ModelDataCollector`:

```
dc = ModelDataCollector(model_name, identifier='default', feature_names=None,  
model_management_account_id='unknown', webservice_name='unknown', model_id='unknown', model_version='unknown')
```

See Class and Parameter details:

Class

NAME	DESCRIPTION
ModelDataCollector	A class in azureml.datacollector namespace. An instance of this class will be used to collect model data. A single scoring file can contain multiple ModelDataCollectors. Each instance should be used for collecting data in one discrete location in the scoring file so that the schema of collected data remains consistent (that is, inputs and prediction)

Parameters

NAME	TYPE	DESCRIPTION
model_name	string	the name of the model which data is being collected for
identifier	string	the location in code that identifies this data, i.e. 'RawInput' or 'Prediction'
feature_names	list of strings	a list of feature names that become the csv header when supplied
model_management_account_id	string	the identifier for the model management account where this model is stored. This is populated automatically when models are operationalized through AML
webservice_name	string	the name of the webservice to which this model is currently deployed. This is populated automatically when models are operationalized through AML
model_id	string	The unique identifier for this model in the context of a model management account. this is populated automatically when models are operationalized through AML
model_version	string	the version number of this model in the context of a model management account. This is populated automatically when models are operationalized through AML

Collecting the model data

You can collect the model data using an instance of the ModelDataCollector created above.

```
dc.collect(input_data, user_correlation_id="")
```

See Method and Parameter details:

Method

NAME	DESCRIPTION
collect	Used to collect the data for a model input or prediction

Parameters

NAME	TYPE	DESCRIPTION
input_data	multiple types	the data to be collected (currently accepts the types list, numpy.array, pandas.DataFrame, pyspark.sql.DataFrame). For dataframe types, if a header exists with feature names, this information is included in the data destination (without needing to explicitly pass feature names in the ModelDataCollector constructor)
user_correlation_id	string	an optional correlation id, which can be provided by the user to correlate this prediction

Python packages for Azure Machine Learning

12/11/2018 • 2 minutes to read • [Edit Online](#)

NOTE

This article is deprecated. Support for earlier versions of this service will end incrementally. [View the support timeline](#). Start using the latest version with this [quickstart](#).

Learn about the proprietary Python packages from Microsoft for Azure Machine Learning (Preview). You can use these libraries and functions in combination with other open-source or third-party packages, but to use the proprietary packages, your Python code must run against a service or on a computer that provides the interpreters.

The Azure Machine Learning packages are **Python pip-installable extensions for Azure Machine Learning** that enable data scientists and AI developers to quickly build and deploy highly accurate machine learning and deep learning models for various domains.

Azure ML Package for Computer Vision

With Azure ML Package for Computer Vision, you can build, fine-tune, and deploy deep learning models for image classification, object detection, and image similarity.

This package is no longer available for download.

Azure ML Package for Forecasting

With Azure ML Package for Forecasting, you can create and deploy time series forecasting models for financial and demand forecasting scenarios.

This package is no longer available for download

Azure ML Package for Text Analytics

With Azure ML Package for Text Analytics, you can build text deep-learning models for text classification, custom entity extraction, and word embedding.

This package is no longer available for download

Build and deploy image classification models with Azure Machine Learning

11/28/2018 • 16 minutes to read • [Edit Online](#)

NOTE

This article is deprecated. Support for earlier versions of this service will end incrementally. [View the support timeline](#). Start using the latest version with this [quickstart](#).

In this article, learn how to use Azure Machine Learning Package for Computer Vision (AMLPCV) to train, test, and deploy an image classification model. For an overview of this package and its detailed reference documentation, [see here](#).

A large number of problems in the computer vision domain can be solved using image classification. These problems include building models that answer questions such as:

- *Is an OBJECT present in the image? For example, "dog", "car", "ship", and so on*
- *What class of eye disease severity is evinced by this patient's retinal scan?*

When building and deploying this model with AMLPCV, you go through the following steps:

1. Dataset Creation
2. Image Visualization and annotation
3. Image Augmentation
4. Deep Neural Network (DNN) Model Definition
5. Classifier Training
6. Evaluation and Visualization
7. Web service Deployment
8. Web service Load Testing

CNTK is used as the deep learning framework, training is performed locally on a GPU powered machine such as the ([Deep learning Data Science VM](#)), and deployment uses the Azure ML Operationalization CLI.

Prerequisites

1. If you don't have an Azure subscription, create a [free account](#) before you begin.
2. The following accounts and application must be set up and installed:
 - An Azure Machine Learning Experimentation account
 - An Azure Machine Learning Model Management account
 - Azure Machine Learning Workbench installedIf these three are not yet created or installed, follow the [Azure Machine Learning Quickstart and Workbench installation](#) article.
3. The Azure Machine Learning Package for Computer Vision must be installed. Learn how to [install this package here](#).

Sample data and notebook

Get the Jupyter notebook

Download the notebook to run the sample described here yourself.

Get the Jupyter notebook

Load the sample data

The following example uses a dataset consisting of 63 tableware images. Each image is labeled as belonging to one of four different classes (bowl, cup, cutlery, plate). The number of images in this example is small so that this sample can be executed quickly. In practice at least 100 images per class should be provided. All images are located at `../sample_data/imgs_recycling/`, in subdirectories called "bowl", "cup", "cutlery", and "plate".



bowl



cup



cutlery



plate

```
import warnings
warnings.filterwarnings("ignore")
import json, numpy as np, os, timeit
from azureml.logging import get_azureml_logger
from imgaug import augmenters
from IPython.display import display
from sklearn import svm
from cvtk import ClassificationDataset, CNTKModel, Context, Splitter, StorageContext
from cvtk.augmentation import augment_dataset
from cvtk.core.classifier import ScikitClassifier
from cvtk.evaluation import ClassificationEvaluation, graph_roc_curve, graph_pr_curve, graph_confusion_matrix
import matplotlib.pyplot as plt

from classification.notebook.ui_utils.ui_annotation import AnnotationUI
from classification.notebook.ui_utils.ui_results_viewer import ResultsUI
from classification.notebook.ui_utils.ui_precision_recall import PrecisionRecallUI

%matplotlib inline

# Disable printing of logging messages
from azuremltkbase.logging import ToolkitLogger
ToolkitLogger.getInstance().setEnabled(False)
```

Create a dataset

Once you have imported the dependencies and set the storage context, you can create the dataset object.

To create that object with Azure Machine Learning Package for Computer Vision, provide the root directory of the images on the local disk. This directory must follow the same general structure as the tableware dataset, that is, contain subdirectories with the actual images:

- root
 - label 1
 - label 2
 - ...
 - label n

Training an image classification model for a different dataset is as easy as changing the root path `dataset_location` in the following code to point at different images.

```

# Root image directory
dataset_location = os.path.abspath("classification/sample_data/imgs_recycling")

dataset_name = 'recycling'
dataset = ClassificationDataset.create_from_dir(dataset_name, dataset_location)
print("Dataset consists of {} images with {} labels.".format(len(dataset.images), len(dataset.labels)))
print("Select information for image 2: name={}, label={}, unique id={}".format(
    dataset.images[2].name, dataset.images[2].label.name, dataset.images[2].storage_id))

```

```

F1 2018-04-23 17:12:57,593 INFO azureml.vision:machine info {"is_dsvm": true, "os_type": "Windows"}
F1 2018-04-23 17:12:57,599 INFO azureml.vision:dataset creating dataset for scenario=classification
Dataset consists of 63 images with 4 labels.
Select information for image 2: name=msft-plastic-bowl20170725152154282.jpg, label=bowl, unique id=3.

```

The dataset object provides functionality to download images using the [Bing Image Search API](#).

Two types of search queries are supported:

- Regular text queries
- Image URL queries

These queries along with the class label must be provided inside a JSON-encoded text file. For example:

```
{
    "bowl": [
        "plastic bowl",
        "../imgs_recycling/bowl"
    ],
    "cup": [
        "plastic cup",
        "../imgs_recycling/cup",
        "http://cdnimg2.webstaurantstore.com/images/products/main/123662/268841/dart-solo-ultra-clear-conex-tp12-12-oz-pet-plastic-cold-cup-1000-case.jpg"
    ],
    "cutlery": [
        "plastic cutlery",
        "../imgs_recycling/cutlery",
        "http://img4.foodservicewarehouse.com/Prd/1900SQ/Fineline_2514-B0.jpg"
    ],
    "plate": [
        "plastic plate",
        "../imgs_recycling/plate"
    ]
}
```

Furthermore, you must explicitly create a Context object to contain the Bing Image Search API key. This requires a Bing Image Search API subscription.

Visualize and annotate images

You can visualize the images and correct labels in the dataset object using the following widget.

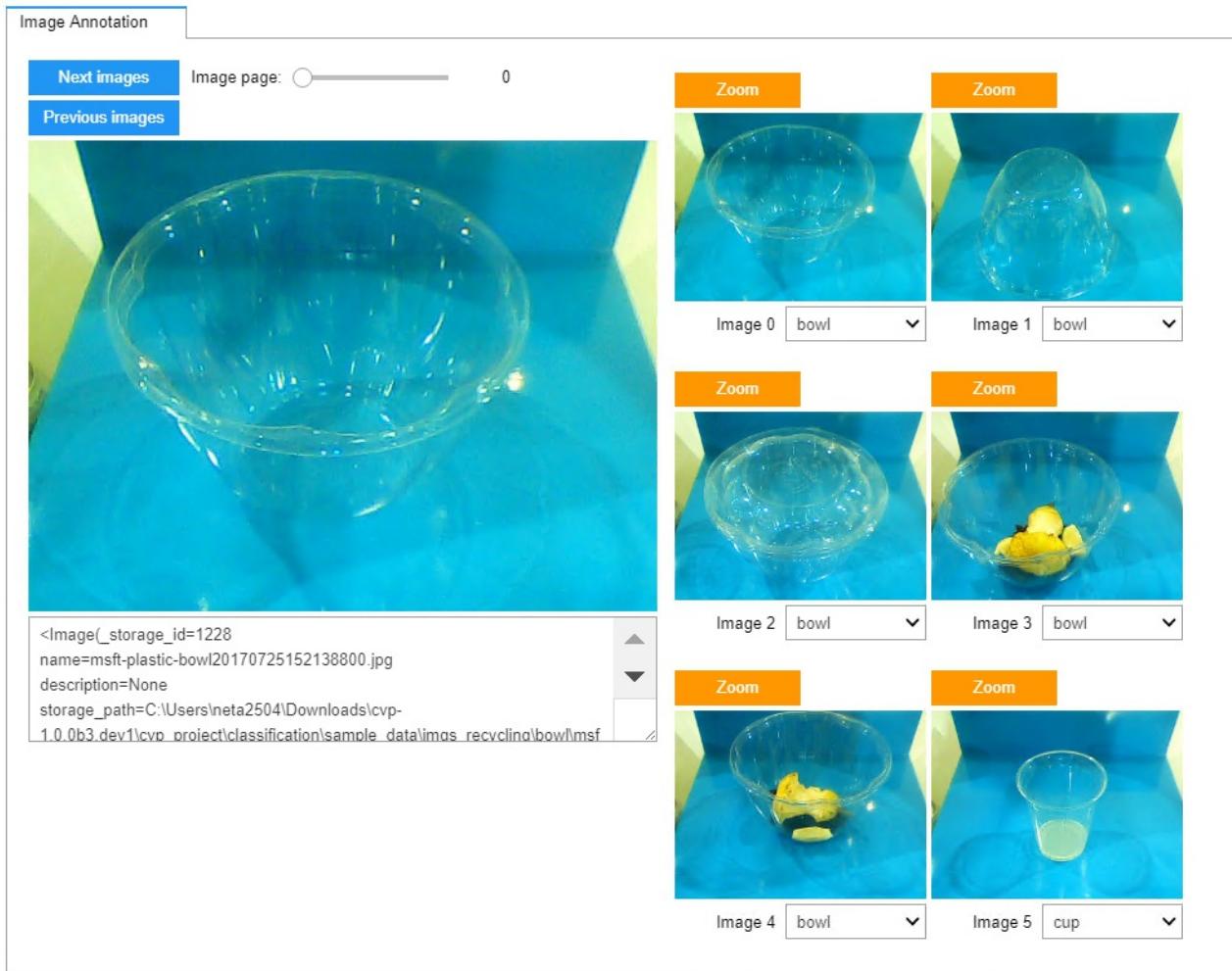
If you encounter the "Widget Javascript not detected" error, run this command to solve it:

```
jupyter nbextension enable --py --sys-prefix widgetsnbextension
```

```

annotation_ui = AnnotationUI(dataset, Context.get_global_context())
display(annotation_ui.ui)

```



Augment images

The [augmentation module](#) provides functionality to augment a dataset object using all the transformations described in the [imgaug](#) library. Image transformations can be grouped in a single pipeline, in which case all transformations in the pipeline are applied simultaneously each image.

If you would like to apply different augmentation steps separately, or in any different manner, you can define multiple pipelines and pass them to the `augment_dataset` function. For more information and examples of image augmentation, see the [imgaug documentation](#).

Adding augmented images to the training set is especially beneficial for small datasets. Since the DNN training process is slower due to the increased number of training images, we recommend you start experimentation without augmentation.

```
# Split the dataset into train and test
train_set_orig, test_set = dataset.split(train_size = 0.66, stratify = "label")
print("Number of training images = {}, test images = {}".format(train_set_orig.size(), test_set.size()))
```

```
F1 2018-04-23 17:13:01,780 INFO azureml.vision:splitter splitting a dataset
F1 2018-04-23 17:13:01,805 INFO azureml.vision:dataset creating dataset for scenario=classification
F1 2018-04-23 17:13:01,809 INFO azureml.vision:dataset creating dataset for scenario=classification
Number of training images = 41, test images = 22.
```

```

augment_train_set = False

if augment_train_set:
    aug_sequence = augmenters.Sequential([
        augmenters.Fliplr(0.5),           # horizontally flip 50% of all images
        augmenters.Crop(percent=(0, 0.1)), # crop images by 0-10% of their height/width
    ])
    train_set = augment_dataset(train_set_orig, [aug_sequence])
    print("Number of original training images = {}, with augmented images included = {}.".format(train_set_orig.size(), train_set.size()))
else:
    train_set = train_set_orig

```

Define DNN models

The following pretrained Deep Neural Network models are supported with this package:

- Resnet-18
- Resnet-34
- Resnet-50
- Resnet-101
- Resnet-152

These DNNs can be used either as classifier, or as featurizer.

More information about the networks can be found [here](#), and a basic introduction to Transfer Learning is [here](#).

The default image classification parameters for this package are 224x224 pixel resolution and a Resnet-18 DNN. These parameters were selected to work well on a wide variety of tasks. Accuracy can often be improved, for example, by increasing the image resolution to 500x500 pixels, and/or selecting a deeper model (Resnet-50). However, changing the parameters can come at a significant increase in training time. See the article on [How to improve accuracy](#).

```

# Default parameters (224 x 224 pixels resolution, Resnet-18)
lr_per_mb = [0.05]*7 + [0.005]*7 + [0.0005]
mb_size = 32
input_resoluton = 224
base_model_name = "ResNet18_ImageNet_CNTK"

# Suggested parameters for 500 x 500 pixels resolution, Resnet-50
# (see in the Appendix "How to improve accuracy", last row in table)
# lr_per_mb = [0.01] * 7 + [0.001] * 7 + [0.0001]
# mb_size = 8
# input_resoluton = 500
# base_model_name = "ResNet50_ImageNet_CNTK"

# Initialize model
dnn_model = CNTKTLModel(train_set.labels,
                        base_model_name=base_model_name,
                        image_dims = (3, input_resoluton, input_resoluton))

```

Successfully downloaded ResNet18_ImageNet_CNTK

Train the classifier

You can choose one of the following methods for the pre-trained DNN.

- **DNN refinement**, which trains the DNN to perform the classification directly. While DNN training is slow, it typically leads to the best results since all network weights can be improved during training to give best accuracy.
- **DNN featurization**, which runs the DNN as-is to obtain a lower-dimensional representation of an image (512, 2048, or 4096 floats). That representation is then used as input to train a separate classifier. Since the DNN is kept unchanged, this approach is much faster compared to DNN refinement, however accuracy is not as good. Nevertheless, training an external classifier such as a linear SVM (as shown in the following code) can provide a strong baseline, and help with understanding the feasibility of a problem.

TensorBoard can be used to visualize the training progress. To activate TensorBoard:

1. Add the parameter `tensorboard_logdir=PATH` as shown in the following code
2. Start the TensorBoard client using the command `tensorboard --logdir=PATH` in a new console.
3. Open a web browser as instructed by TensorBoard, which by default is localhost:6006.

```
# Train either the DNN or a SVM as classifier
classifier_name = "dnn"

if classifier_name.lower() == "dnn":
    dnn_model.train(train_set, lr_per_mb = lr_per_mb, mb_size = mb_size, eval_dataset=test_set) #,
    tensorboard_logdir=r"tensorboard"
    classifier = dnn_model
elif classifier_name.lower() == "svm":
    learner = svm.LinearSVC(C=1.0, class_weight='balanced', verbose=0)
    classifier = ScikitClassifier(dnn_model, learner = learner)
    classifier.train(train_set)
else:
    raise Exception("Classifier unknown: " + classifier)
```

```

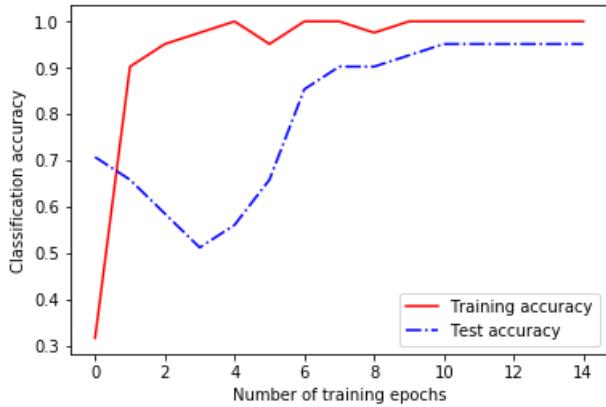
F1 2018-04-23 17:13:28,238 INFO azureml.vision:Fit starting in experiment 1541466320
F1 2018-04-23 17:13:28,239 INFO azureml.vision:model starting training for scenario=classification
<class 'int'>
1 worker
Training transfer learning model for 15 epochs (epoch_size = 41).
non-distributed mode
Training 15741700 parameters in 53 parameter tensors.
Training 15741700 parameters in 53 parameter tensors.
Learning rate per minibatch: 0.05
Momentum per minibatch: 0.9
PROGRESS: 0.00%
Finished Epoch[1 of 15]: [Training] loss = 2.820586 * 41, metric = 68.29% * 41 5.738s ( 7.1 samples/s);
Evaluation Set Error :: 29.27%
Finished Epoch[2 of 15]: [Training] loss = 0.286728 * 41, metric = 9.76% * 41 0.752s ( 54.5 samples/s);
Evaluation Set Error :: 34.15%
Finished Epoch[3 of 15]: [Training] loss = 0.206938 * 41, metric = 4.88% * 41 0.688s ( 59.6 samples/s);
Evaluation Set Error :: 41.46%
Finished Epoch[4 of 15]: [Training] loss = 0.098931 * 41, metric = 2.44% * 41 0.785s ( 52.2 samples/s);
Evaluation Set Error :: 48.78%
Finished Epoch[5 of 15]: [Training] loss = 0.046547 * 41, metric = 0.00% * 41 0.724s ( 56.6 samples/s);
Evaluation Set Error :: 43.90%
Finished Epoch[6 of 15]: [Training] loss = 0.059709 * 41, metric = 4.88% * 41 0.636s ( 64.5 samples/s);
Evaluation Set Error :: 34.15%
Finished Epoch[7 of 15]: [Training] loss = 0.005817 * 41, metric = 0.00% * 41 0.710s ( 57.7 samples/s);
Evaluation Set Error :: 14.63%
Learning rate per minibatch: 0.005
Finished Epoch[8 of 15]: [Training] loss = 0.014917 * 41, metric = 0.00% * 41 0.649s ( 63.2 samples/s);
Evaluation Set Error :: 9.76%
Finished Epoch[9 of 15]: [Training] loss = 0.040539 * 41, metric = 2.44% * 41 0.777s ( 52.8 samples/s);
Evaluation Set Error :: 9.76%
Finished Epoch[10 of 15]: [Training] loss = 0.024606 * 41, metric = 0.00% * 41 0.626s ( 65.5 samples/s);
Evaluation Set Error :: 7.32%
PROGRESS: 0.00%
Finished Epoch[11 of 15]: [Training] loss = 0.004225 * 41, metric = 0.00% * 41 0.656s ( 62.5 samples/s);
Evaluation Set Error :: 4.88%
Finished Epoch[12 of 15]: [Training] loss = 0.004364 * 41, metric = 0.00% * 41 0.702s ( 58.4 samples/s);
Evaluation Set Error :: 4.88%
Finished Epoch[13 of 15]: [Training] loss = 0.007974 * 41, metric = 0.00% * 41 0.721s ( 56.9 samples/s);
Evaluation Set Error :: 4.88%
Finished Epoch[14 of 15]: [Training] loss = 0.000655 * 41, metric = 0.00% * 41 0.711s ( 57.7 samples/s);
Evaluation Set Error :: 4.88%
Learning rate per minibatch: 0.0005
Finished Epoch[15 of 15]: [Training] loss = 0.024865 * 41, metric = 0.00% * 41 0.688s ( 59.6 samples/s);
Evaluation Set Error :: 4.88%
Stored trained model at ../../cvtk_output\model_trained\ImageClassification.model
F1 2018-04-23 17:13:45,097 INFO azureml.vision:Fit finished in experiment 1541466320

```

```

# Plot how the training and test accuracy increases during gradient descent.
if classifier_name == "dnn":
    [train_accs, test_accs, epoch_numbers] = classifier.train_eval_accs
    plt.xlabel("Number of training epochs")
    plt.ylabel("Classification accuracy")
    train_plot = plt.plot(epoch_numbers, train_accs, 'r-', label = "Training accuracy")
    test_plot = plt.plot(epoch_numbers, test_accs, 'b--', label = "Test accuracy")
    plt.legend()

```



Evaluate and visualize model performance

You can evaluate the performance of the trained model on an independent test dataset using the evaluation module. Some of the evaluation metrics it computes include:

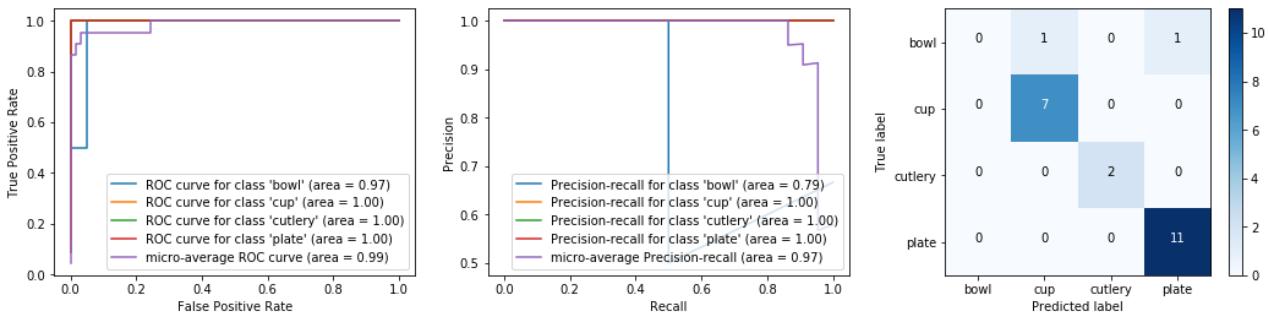
- Accuracy (by default class-averaged)
- PR curve
- ROC curve
- Area-under-curve
- Confusion matrix

```
# Run the classifier on all test set images
ce = ClassificationEvaluation(classifier, test_set, minibatch_size = mb_size)

# Compute Accuracy and the confusion matrix
acc = ce.compute_accuracy()
print("Accuracy = {:.2f}%".format(100*acc))
cm = ce.compute_confusion_matrix()
print("Confusion matrix = \n{}".format(cm))

# Show PR curve, ROC curve, and confusion matrix
fig, ((ax1, ax2, ax3)) = plt.subplots(1,3)
fig.set_size_inches(18, 4)
graph_roc_curve(ce, ax=ax1)
graph_pr_curve(ce, ax=ax2)
graph_confusion_matrix(ce, ax=ax3)
plt.show()
```

```
F1 2018-04-23 17:14:37,449 INFO azureml.vision:evaluation doing evaluation for scenario=classification
F1 2018-04-23 17:14:37,450 INFO azureml.vision:model scoring dataset for scenario=classification
Accuracy = 95.45%
Confusion matrix =
[[ 0  1  0  1]
 [ 0  7  0  0]
 [ 0  0  2  0]
 [ 0  0  0 11]]
```



```
# Results viewer UI
labels = [l.name for l in dataset.labels]
pred_scores = ce.scores #classification scores for all images and all classes
pred_labels = [labels[i] for i in np.argmax(pred_scores, axis=1)]

results_ui = ResultsUI(test_set, Context.get_global_context(), pred_scores, pred_labels)
display(results_ui.ui)
```

Results viewer

Image -1
Image +1
0



Filters (use Image +/- buttons for navigation):

Correct classifications
 Incorrect classifications

Ground truth:

Label: plate

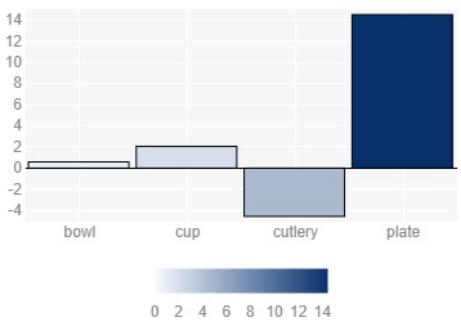
Prediction:

Label: plate
Score: 14.506685256958008

Image info:

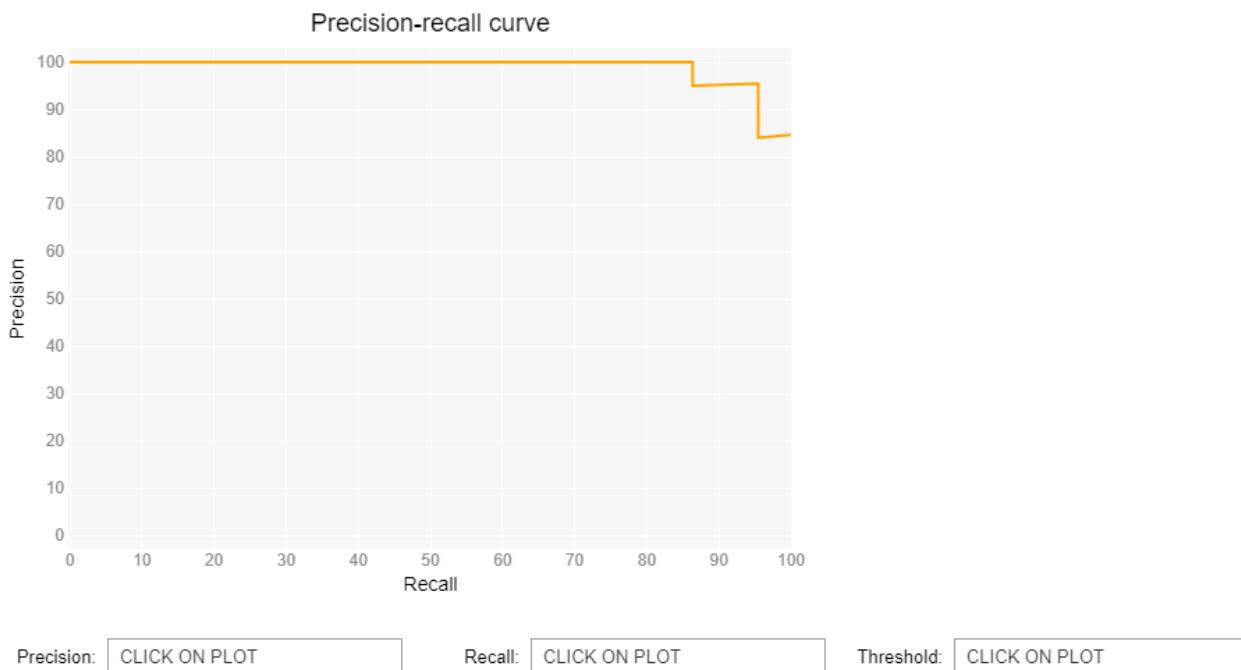
Index: 0
Name: msft-paper-plate20170725151127
StoragePath: C:\Users\neta2504\Downloads\cv

Classification scores:



Class	Score
bowl	~0.5
cup	~2.0
cutlery	~-3.0
plate	~14.0

```
# Precision / recall curve UI
precisions, recalls, thresholds = ce.compute_precision_recall_curve()
thresholds = list(thresholds)
thresholds.append(thresholds[-1])
pr_ui = PrecisionRecallUI(100*precisions[::-1], 100*recalls[::-1], thresholds[::-1])
display(pr_ui.ui)
```



Operationalization: deploy and consume

Operationalization is the process of publishing models and code as web services and the consumption of these services to produce business results.

Once your model is trained, you can deploy that model as a web service for consumption using [Azure Machine Learning CLI](#). Your models can be deployed to your local machine or Azure Container Service (ACS) cluster. Using ACS, you can scale your web service manually or use the autoscaling functionality.

Sign in with Azure CLI

Using an [Azure](#) account with a valid subscription, log in using the following CLI command:

```
az login
```

- To switch to another Azure subscription, use the command:

```
az account set --subscription [your subscription name]
```

- To see the current model management account, use the command:

```
az ml account modelmanagement show
```

Create and set your cluster deployment environment

You only need to set your deployment environment once. If you don't have one yet, set up your deployment environment now using [these instructions](#).

To see your active deployment environment, use the following CLI command:

```
az ml env show
```

Sample Azure CLI command to create and set deployment environment

```
az provider register -n Microsoft.MachineLearningCompute
az provider register -n Microsoft.ContainerRegistry
az provider register -n Microsoft.ContainerService
az ml env setup --cluster -n [your environment name] -l [Azure region e.g. westcentralus] [-g [resource group]]
az ml env set -n [environment name] -g [resource group]
az ml env cluster
```

Manage web services and deployments

The following APIs can be used to deploy models as web services, manage those web services, and manage deployments.

TASK	API
Create deployment object	<pre>deploy_obj = AMLDeployment(deployment_name=deployment_name, associated_DNNModel=dnn_model, aml_env="cluster")</pre>
Deploy web service	<pre>deploy_obj.deploy()</pre>
Score image	<pre>deploy_obj.score_image(local_image_path_or_image_url)</pre>
Delete web service	<pre>deploy_obj.delete()</pre>
Build docker image without web service	<pre>deploy_obj.build_docker_image()</pre>
List existing deployment	<pre>AMLDisclosure.list_deployment()</pre>
Delete if the service exists with the deployment name	<pre>AMLDisclosure.delete_if_service_exist(deployment_name)</pre>

API documentation: Consult the [package reference documentation](#) for the detailed reference for each module and class.

CLI reference: For more advanced operations related to deployment, refer to the [model management CLI reference](#).

Deployment management in Azure portal: You can track and manage your deployments in the [Azure portal](#). From the Azure portal, find your Machine Learning Model Management account page using its name. Then go to the Model Management account page > Model Management > Services.

```
# ##### OPTIONAL#####
# Interactive CLI setup helper, including model management account and deployment environment.
# If you haven't setup your CLI before or if you want to change your CLI settings, you can use this block to
# help you interactively.
#
# UNCOMMENT THE FOLLOWING LINES IF YOU HAVE NOT CREATED OR SET THE MODEL MANAGEMENT ACCOUNT AND DEPLOYMENT
# ENVIRONMENT
#
# from azuremltkbase.deployment import CliSetup
# CliSetup().run()
```

```
# # Optional. Persist your model on disk and reuse it later for deployment.
# from cvtk import TFFasterRCNN, Context
# import os
# save_model_path = os.path.join(Context.get_global_context().storage.persistent_path,
# "saved_classifier.model")
# # Save model to disk
# dnn_model.serialize(save_model_path)
# # Load model from disk
# dnn_model = CNTKTLModel.deserialize(save_model_path)
```

```

from cvtk.operationalization import AMLDeployment

# set deployment name
deployment_name = "wsdeployment"

# Optional Azure Machine Learning deployment cluster name (environment name) and resource group name
# If you don't provide here. It will use the current deployment environment (you can check with CLI command
#"az ml env show").
azureml_rscgroup = "<resource group>"
cluster_name = "<cluster name>"

# If you provide the cluster information, it will use the provided cluster to deploy.
# Example: deploy_obj = AMLDeployment(deployment_name=deployment_name, associated_DNNModel=dnn_model,
#                                     aml_env="cluster", cluster_name=cluster_name, resource_group=azureml_rscgroup,
#                                     replicas=1)

# Create deployment object
deploy_obj = AMLDeployment(deployment_name=deployment_name, aml_env="cluster", associated_DNNModel=dnn_model,
replicas=1)

# Check if the deployment name exists, if yes remove it first.
if deploy_obj.is_existing_service():
    AMLDeployment.delete_if_service_exist(deployment_name)

# Create the web service
print("Deploying to Azure cluster...")
deploy_obj.deploy()
print("Deployment DONE")

```

Consume the web service

Once you deploy the model as a web service, you can score images with the web service using one of these methods:

- Score the web service directly with the deployment object using `deploy_obj.score_image(image_path_or_url)`
- Use the Service endpoint URL and Service key (None for local deployment) with:
`AMLDeployment.score_existing_service_with_image(image_path_or_url, service_endpoint_url,`
`service_key=None)`
- Form your HTTP requests directly to score the web service endpoint. This option is for advanced users.

Score with existing deployment object

```
deploy_obj.score_image(image_path_or_url)
```

```

# Score with existing deployment object

# Score local image with file path
print("Score local image with file path")
image_path_or_url = test_set.images[0].storage_path
print("Image source:",image_path_or_url)
serialized_result_in_json = deploy_obj.score_image(image_path_or_url, image_resize_dims=[224,224])
print("serialized_result_in_json:", serialized_result_in_json)

# Score image url and remove image resizing
print("Score image url")
image_path_or_url = "https://cvtkdata.blob.core.windows.net/publicimages/microsoft_logo.jpg"
print("Image source:",image_path_or_url)
serialized_result_in_json = deploy_obj.score_image(image_path_or_url)
print("serialized_result_in_json:", serialized_result_in_json)

# Score image url with added parameters. Add softmax to score.
print("Score image url with added parameters. Add softmax to score")
from cvtk.utils.constants import ClassificationRESTApiParameters
image_path_or_url = "https://cvtkdata.blob.core.windows.net/publicimages/microsoft_logo.jpg"
print("Image source:",image_path_or_url)
serialized_result_in_json = deploy_obj.score_image(image_path_or_url, image_resize_dims=[224,224], parameters={ClassificationRESTApiParameters.ADD_SOFTMAX:True})
print("serialized_result_in_json:", serialized_result_in_json)

```

```

# Time image scoring
import timeit

for img_index, img_obj in enumerate(test_set.images[:10]):
    print("Calling API for image {} of {}: {}".format(img_index, len(test_set.images), img_obj.name))
    tic = timeit.default_timer()
    return_json = deploy_obj.score_image(img_obj.storage_path, image_resize_dims=[224,224])
    print("Time for API call: {:.2f} seconds".format(timeit.default_timer() - tic))
    print(return_json)

```

Score with service endpoint url and service key

```
AMLDisclosure.score_existing_service_with_image(image_path_or_url, service_endpoint_url, service_key=None)
```

```

# Import related classes and functions
from cvtk.operationalization import AMLDeployment

service_endpoint_url = "" # please replace with your own service url
service_key = "" # please replace with your own service key
# score local image with file path
image_path_or_url = test_set.images[0].storage_path
print("Image source:",image_path_or_url)
serialized_result_in_json =
AMLDisclosure.score_existing_service_with_image(image_path_or_url,service_endpoint_url, service_key =
service_key)
print("serialized_result_in_json:", serialized_result_in_json)

# score image url
image_path_or_url = "https://cvtkdata.blob.core.windows.net/publicimages/microsoft_logo.jpg"
print("Image source:",image_path_or_url)
serialized_result_in_json =
AMLDisclosure.score_existing_service_with_image(image_path_or_url,service_endpoint_url, service_key =
service_key, image_resize_dims=[224,224])
print("serialized_result_in_json:", serialized_result_in_json)

```

Score endpoint with http request directly

The following example code forms the HTTP request directly in Python. However, you can do it in other

programming languages.

```
def score_image_list_with_http(images, service_endpoint_url, service_key=None, parameters={}):
    """Score image list with http request

    Args:
        images(list): list of (input image file path, base64 image string, url or buffer)
        service_endpoint_url(str): endpoint url
        service_key(str): service key, None for local deployment.
        parameters(dict): service additional parameters in dictionary

    Returns:
        result (list): list of serialized result
    """
    import requests
    from io import BytesIO
    import base64
    routing_id = ""

    if service_key is None:
        headers = {'Content-Type': 'application/json',
                   'X-Marathon-App-Id': routing_id}
    else:
        headers = {'Content-Type': 'application/json',
                   'Authorization': ('Bearer ' + service_key), 'X-Marathon-App-Id': routing_id}
    payload = []
    for image in images:
        encoded = None
        # read image
        with open(image, 'rb') as f:
            image_buffer = BytesIO(f.read()) ## Getting an image file represented as a BytesIO object
            # convert your image to base64 string
            encoded = base64.b64encode(image_buffer.getvalue())
        image_request = {"image_in_base64": "{0}".format(encoded), "parameters": parameters}
        payload.append(image_request)
    body = json.dumps(payload)
    r = requests.post(service_endpoint_url, data=body, headers=headers)
    try:
        result = json.loads(r.text)
    except:
        raise ValueError("Incorrect output format. Result cant not be parsed: " + r.text)
    return result

# Test with images
images = [test_set.images[0].storage_path, test_set.images[1].storage_path] # A list of local image files
score_image_list_with_http(images, service_endpoint_url, service_key)
```

Parse serialized result from web service

The output from the web service is a JSON string. You can parse this JSON string with different DNN model classes.

```
image_path_or_url = test_set.images[0].storage_path
print("Image source:",image_path_or_url)
serialized_result_in_json = deploy_obj.score_image(image_path_or_url, image_resize_dims=[224,224])
print("serialized_result_in_json:", serialized_result_in_json)
```

```
# Parse result from json string
import numpy as np
parsed_result = CNTKTLModel.parse_serialized_result(serialized_result_in_json)
print("Parsed result:", parsed_result)
# Map result to image class
class_index = np.argmax(np.array(parsed_result))
print("Class index:", class_index)
dnn_model.class_map
print("Class label:", dnn_model.class_map[class_index])
```

Next steps

Learn more about Azure Machine Learning Package for Computer Vision in these articles:

- Learn how to [improve the accuracy of this model](#).
- Read the [package overview](#).
- Explore the [reference documentation](#) for this package.
- Learn about [other Python packages for Azure Machine Learning](#).

Build and deploy object detection models with Azure Machine Learning

11/13/2018 • 14 minutes to read • [Edit Online](#)

NOTE

This article is deprecated. Support for earlier versions of this service will end incrementally. [View the support timeline](#). Start using the latest version with this [quickstart](#).

In this article, learn how to use **Azure Machine Learning Package for Computer Vision** to train, test, and deploy a [Faster R-CNN](#) object detection model.

A large number of problems in the computer vision domain can be solved using object detection. These problems include building models that find a variable number of objects on an image.

When building and deploying this model with this package, you go through the following steps:

1. Dataset Creation
2. Deep Neural Network (DNN) Model Definition
3. Model Training
4. Evaluation and Visualization
5. Web service Deployment
6. Web service Load Testing

In this example, TensorFlow is used as the deep learning framework, training is performed locally on a GPU powered machine such as the [Deep learning Data Science VM](#), and deployment uses the Azure ML Operationalization CLI.

Consult the [package reference documentation](#) for the detailed reference for each module and class.

Prerequisites

1. If you don't have an Azure subscription, create a [free account](#) before you begin.
2. The following accounts and application must be set up and installed:
 - An Azure Machine Learning Experimentation account
 - An Azure Machine Learning Model Management account
 - Azure Machine Learning Workbench installedIf these three are not yet created or installed, follow the [Azure Machine Learning Quickstart and Workbench installation](#) article.
3. The Azure Machine Learning Package for Computer Vision must be installed. Learn how to [install this package here](#).

Sample data and notebook

Get the Jupyter notebook

Download the notebook to run the sample described here yourself.

[Get the Jupyter notebook](#)

Load the sample data

For this demo, a dataset of grocery items inside refrigerators is provided, consisting of 30 images and 8 classes (eggBox, joghurt, ketchup, mushroom, mustard, orange, squash, and water). For each jpg image, there's an annotation xml-file with similar name.

The following figure shows the recommended folder structure.

Folder	Description
/	Root directory
/Annotations/	Annotations folder
/Annotations/1.xml	annotations
/Annotations/2.xml	annotations
/Annotations/3.xml	annotations
/JPEGImages/	Images folder
/JPEGImages/1.jpg	Image
/JPEGImages/2.jpg	Image
/JPEGImages/3.jpg	Image

Image Annotation

Annotated object locations are required to train and evaluate an object detector. [LabellImg](#) is an open source annotation tool that can be used to annotate images. LabellImg writes an xml-file per image in Pascal-VOC format, which can be read by this package.

```
import warnings
warnings.filterwarnings("ignore")
import os, time
from cvtk.core import Context, ObjectDetectionDataset, TFFasterRCNN
from cvtk.evaluation import DetectionEvaluation
from cvtk.evaluation.evaluation_utils import graph_error_counts
from cvtk.utils import detection_utils

# Disable printing of logging messages
from azuremltkbase.logging import ToolkitLogger
ToolkitLogger.getInstance().setEnabled(False)

from matplotlib import pyplot as plt
# Display the images
%matplotlib inline
```

Create a dataset

Create a CVTK dataset that consists of a set of images, with their respective bounding box annotations. In this example, the refrigerator images that are provided in the "../sample_data/foods/training" folder are used. Only JPEG images are supported.

```

image_folder = "detection/sample_data/foods/train"
data_train = ObjectDetectionDataset.create_from_dir(dataset_name='training_dataset', data_dir=image_folder,
                                                    annotations_dir="Annotations",
                                                    image_subdirectory='JPEGImages')

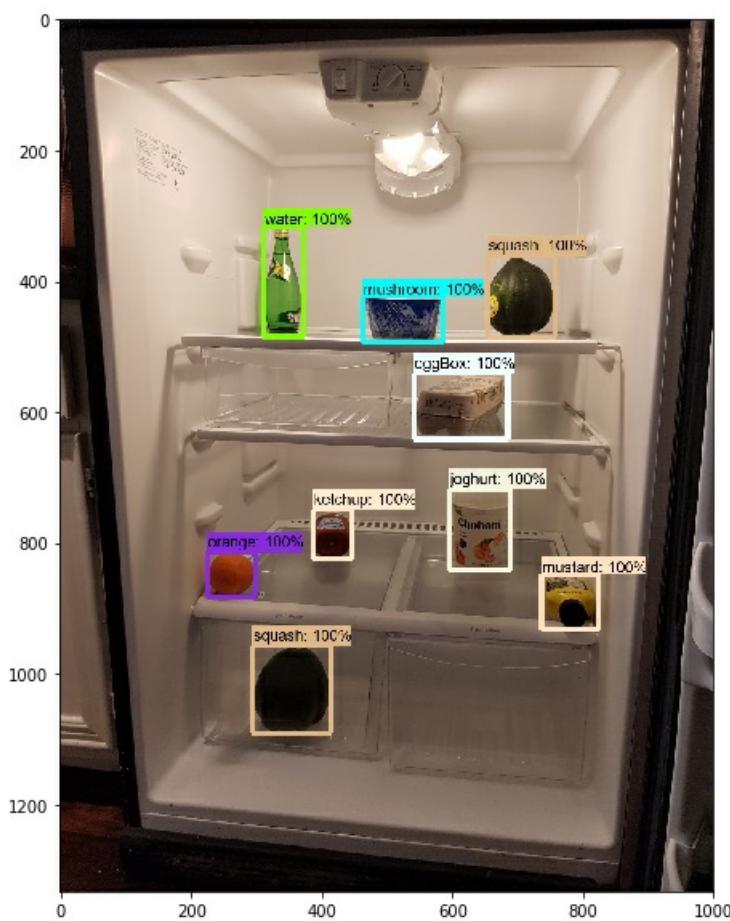
# Show some statistics of the training image, and also give one example of the ground truth rectangle
# annotations
data_train.print_info()
_ = data_train.images[2].visualize_bounding_boxes(image_size = (10,10))

```

```

F1 2018-05-25 23:12:21,727 INFO azureml.vision:machine info {"is_dsvm": true, "os_type": "Windows"}
F1 2018-05-25 23:12:21,733 INFO azureml.vision:dataset creating dataset for scenario=detection
Dataset name: training_dataset
Total classes: 8, total images: 25
Label-wise object counts:
    Label eggBox: 20 objects
    Label joghurt: 20 objects
    Label ketchup: 20 objects
    Label mushroom: 20 objects
    Label mustard: 20 objects
    Label orange: 20 objects
    Label squash: 40 objects
    Label water: 20 objects
Bounding box width and height distribution:
    Bounding box widths 0/5/25/50/75/95/100-percentile: 54/61/79/117/133/165/311 pixels
    Bounding box heights 0/5/25/50/75/95/100-percentile: 48/58/75/124/142/170/212 pixels

```



Define a model

In this example, the Faster R-CNN model is used. Various parameters can be provided when defining this model. The meaning of these parameters, as well as the parameters used for training (see next section) can be found in either CVTK's API docs, or on the [Tensorflow object detection website](#). More information about Faster R-CNN

model can be found at [this link](#). This model is based on Fast R-CNN and more information about it can be found [here](#).

```
score_threshold = 0.0      # Threshold on the detection score, use to discard lower-confidence detections.
max_total_detections = 300 # Maximum number of detections. A high value slows down training but might increase accuracy.
my_detector = TFFasterRCNN(labels=data_train.labels,
                            score_threshold=score_threshold,
                            max_total_detections=max_total_detections)
```

Train the model

The COCO-trained Faster R-CNN model with ResNet50 is used as the starting point for training.

To train the detector, the number of training steps in the code is set to 350, so that training runs more quickly (~5 minutes with GPU). In practice, set it to at least 10 times the number of images in the training set.

In this example, the number of detector training steps is set to 350 for speedy training. However, in practice, a good rule of thumb is to set the steps to 10 or more times the number of images in the training set.

Two key parameters for training are:

- Number of steps to train the model, represented by the num_steps argument. Each step trains the model with a minibatch of batch size one.
- Learning rate(s), which can be set by initial_learning_rate

```
print("tensorboard --logdir={}".format(my_detector.train_dir))

# to get good results, use a larger value for num_steps, e.g., 5000.
num_steps = 350
learning_rate = 0.001 # learning rate

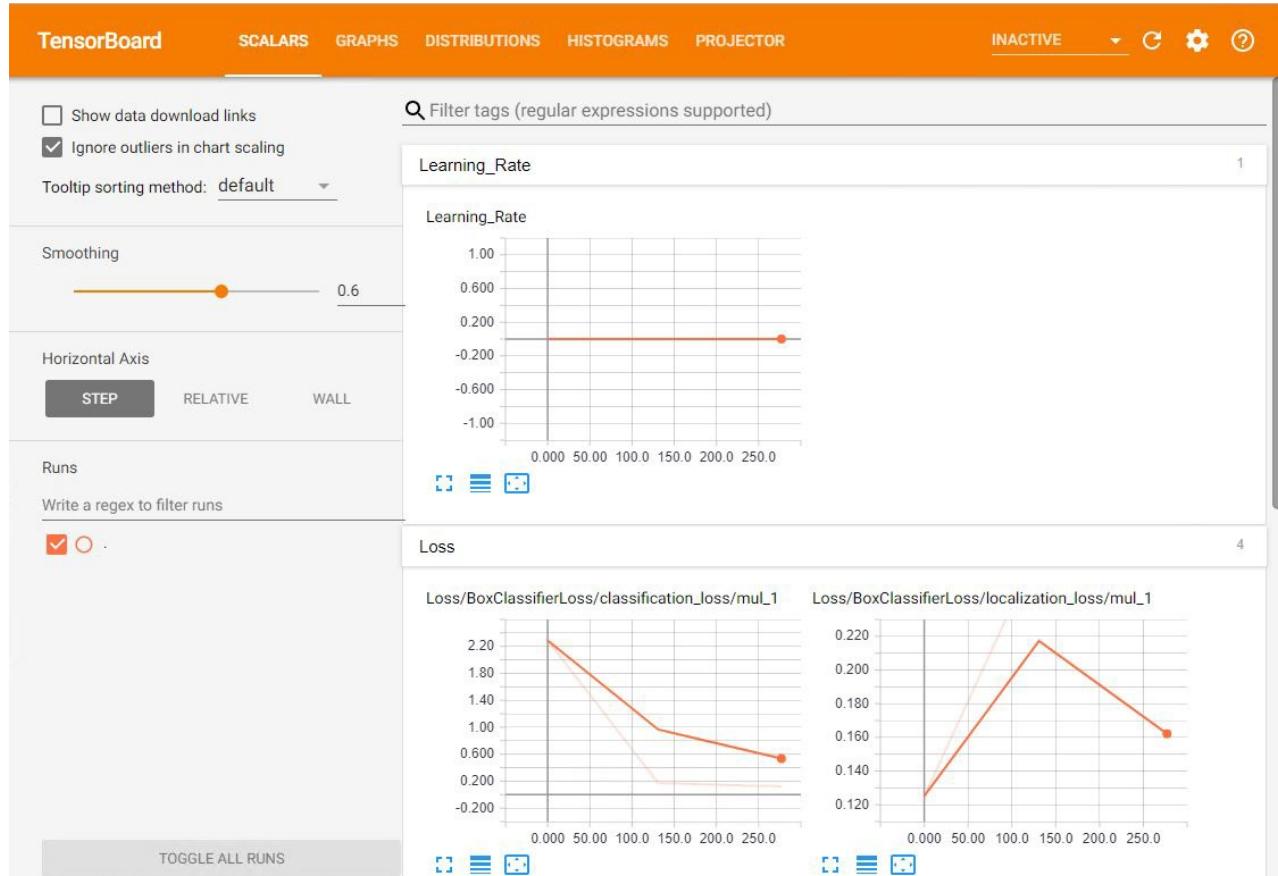
start_train = time.time()
my_detector.train(dataset=data_train, num_steps=num_steps,
                  initial_learning_rate=learning_rate)
end_train = time.time()
print(end_train-start_train)
```

```
tensorboard --
logdir=C:\Users\lixun\Desktop\AutoDL\CVTK\Src\API\cvtk_output\temp_faster_rcnn_resnet50\models\train
F1 2018-05-25 23:12:22,764 INFO azureml.vision:Fit starting in experiment 1125722225
F1 2018-05-25 23:12:22,767 INFO azureml.vision:model starting trainging for scenario=detection
Using existing checkpoint file that's saved at
'C:\Users\lixun\Desktop\AutoDL\CVTK\Src\API\cvtk_output\models\detection\faster_rcnn_resnet50_coco_2018_01_28\model.ckpt.index'.
TFRecords creation started.
F1 2018-05-25 23:12:22,773 INFO On image 0 of 25
TFRecords creation completed.
Training started.
Training prograssing: step 0 ...
Training prograssing: step 100 ...
Training prograssing: step 200 ...
Training prograssing: step 300 ...
F1 2018-05-25 23:18:02,730 INFO Graph Rewriter optimizations enabled
Converted 275 variables to const ops.
F1 2018-05-25 23:18:10,722 INFO 2953 ops in the final graph.
F1 2018-05-25 23:18:24,244 INFO azureml.vision:Fit finished in experiment 1125722225
Training completed.
361.604615688324
```

TensorBoard can be used to visualize the training progress. TensorBoard events are located in the folder specified by the model object's train_dir attribute. To view TensorBoard, follow these steps:

1. Copy the printout that starts with 'tensorboard --logdir' to a command line and run it.
2. Copy the returned URL from the command line to a web browser to view the TensorBoard.

The TensorBoard should look like the following screenshot. It takes a few moments for the training folder to be populated. So if TensorBoard does not show up correctly the first time try repeating steps 1-2.



Evaluate the model

The 'evaluate' method is used to evaluate the model. This function requires an ObjectDetectionDataset object as an input. The evaluation dataset can be created using the same function as the one used for the training dataset. The supported metric is Average Precision as defined for the [PASCAL VOC Challenge](#).

```
image_folder = "detection/sample_data/foods/test"
data_val = ObjectDetectionDataset.create_from_dir(dataset_name='val_dataset', data_dir=image_folder)
eval_result = my_detector.evaluate(dataset=data_val)
```

```

F1 2018-05-25 23:18:24,253 INFO azureml.vision:dataset creating dataset for scenario=detection
F1 2018-05-25 23:18:24,286 INFO On image 0 of 5
F1 2018-05-25 23:18:29,300 INFO Starting evaluation at 2018-05-26-03:18:29
F1 2018-05-25 23:18:32,403 INFO Creating detection visualizations.
F1 2018-05-25 23:18:33,158 INFO Detection visualizations written to summary with tag image-0.
F1 2018-05-25 23:18:33,518 INFO Creating detection visualizations.
F1 2018-05-25 23:18:34,342 INFO Detection visualizations written to summary with tag image-1.
F1 2018-05-25 23:18:34,714 INFO Creating detection visualizations.
F1 2018-05-25 23:18:35,470 INFO Detection visualizations written to summary with tag image-2.
F1 2018-05-25 23:18:35,835 INFO Creating detection visualizations.
F1 2018-05-25 23:18:36,654 INFO Detection visualizations written to summary with tag image-3.
F1 2018-05-25 23:18:37,010 INFO Creating detection visualizations.
F1 2018-05-25 23:18:37,798 INFO Detection visualizations written to summary with tag image-4.
F1 2018-05-25 23:18:37,804 INFO Running eval batches done.
F1 2018-05-25 23:18:37,805 INFO # success: 5
F1 2018-05-25 23:18:37,806 INFO # skipped: 0
F1 2018-05-25 23:18:38,119 INFO Writing metrics to tf summary.
F1 2018-05-25 23:18:38,121 INFO PASCAL/PerformanceByCategory/AP@0.5IOU/eggBox: 1.000000
F1 2018-05-25 23:18:38,205 INFO PASCAL/PerformanceByCategory/AP@0.5IOU/joghurt: 0.942857
F1 2018-05-25 23:18:38,206 INFO PASCAL/PerformanceByCategory/AP@0.5IOU/ketchup: 1.000000
F1 2018-05-25 23:18:38,207 INFO PASCAL/PerformanceByCategory/AP@0.5IOU/mushroom: 1.000000
F1 2018-05-25 23:18:38,208 INFO PASCAL/PerformanceByCategory/AP@0.5IOU/mustard: 1.000000
F1 2018-05-25 23:18:38,209 INFO PASCAL/PerformanceByCategory/AP@0.5IOU/orange: 1.000000
F1 2018-05-25 23:18:38,210 INFO PASCAL/PerformanceByCategory/AP@0.5IOU/squash: 1.000000
F1 2018-05-25 23:18:38,211 INFO PASCAL/PerformanceByCategory/AP@0.5IOU/water: 1.000000
F1 2018-05-25 23:18:38,211 INFO PASCAL/Precision/mAP@0.5IOU: 0.992857
F1 2018-05-25 23:18:38,253 INFO Metrics written to tf summary.
F1 2018-05-25 23:18:38,254 INFO Finished evaluation!

```

The evaluation results can be printed out in a clean format.

```

# print out the performance metric values
for label_obj in data_train.labels:
    label = label_obj.name
    key = 'PASCAL/PerformanceByCategory/AP@0.5IOU/' + label
    print('{0: <15}: {1: <3}'.format(label, round(eval_result[key], 2)))
print('{0: <15}: {1: <3}'.format("overall:", round(eval_result['PASCAL/Precision/mAP@0.5IOU'], 2)))

```

joghurt	:	0.94
squash	:	1.0
mushroom	:	1.0
eggBox	:	1.0
ketchup	:	1.0
mustard	:	1.0
water	:	1.0
orange	:	1.0
overall:	:	0.99

Similarly, you can compute the accuracy of the model on the training set. Doing this helps make sure training converged to a good solution. The accuracy on the training set after successful training is often close to 100%.

Evaluation results can also be viewed from TensorBoard, including mAP values and images with predicted bounding boxes. Copy the printout from the following code into a command line window to start the TensorBoard client. Here a port value 8008 is used to avoid conflict with the default value of 6006, which was using for viewing training status.

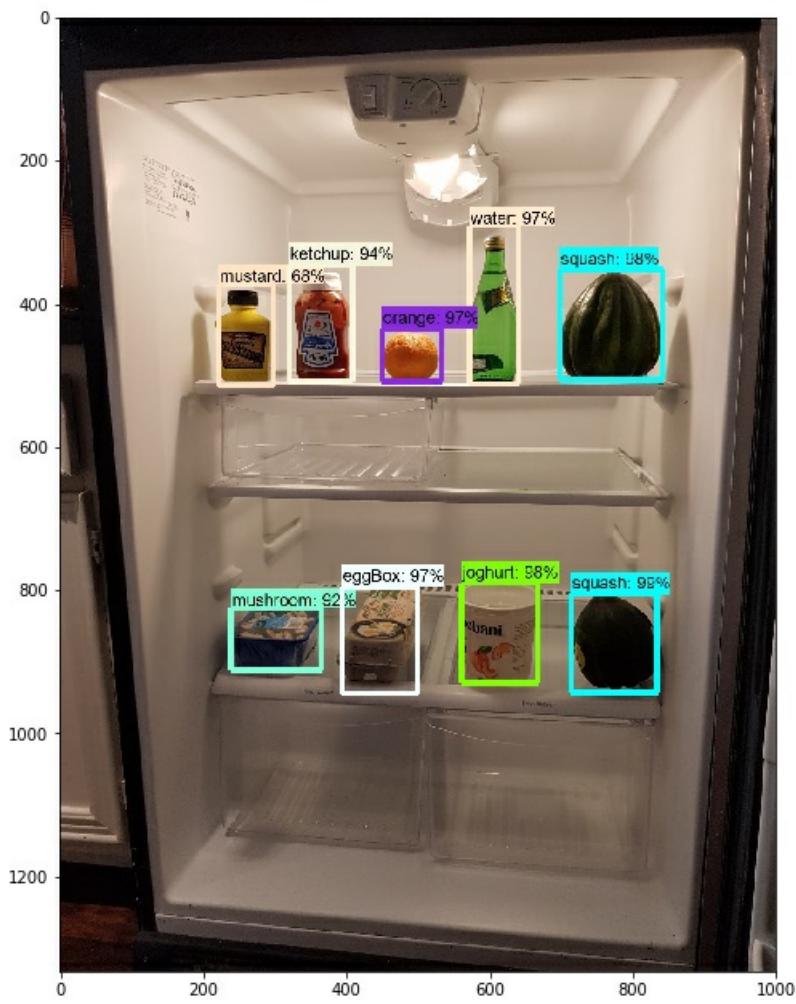
```
print("tensorboard --logdir={} --port=8008".format(my_detector.eval_dir))
```

```
tensorboard --  
logdir=C:\Users\lixun\Desktop\AutoDL\CVTK\Src\API\cvtk_output\temp_faster_rcnn_resnet50\models\eval --port=8008
```

Score an image

Once you're satisfied with the performance of the trained model, the model object's 'score' function can be used to score new images. The returned scores can be visualized with the 'visualize' function .

```
image_path = data_val.images[1].storage_path  
detections_dict = my_detector.score(image_path)  
path_save = "./scored_images/scored_image_preloaded.jpg"  
ax = detection_utils.visualize(image_path, detections_dict, image_size=(8, 12))  
path_save_dir = os.path.dirname(os.path.abspath(path_save))  
os.makedirs(path_save_dir, exist_ok=True)  
ax.get_figure().savefig(path_save)
```



Save the model

The trained model can be saved to disk, and loaded back into memory, as shown in the following code examples.

```
save_model_path = "./frozen_model/faster_rcnn.model"  
my_detector.save(save_model_path)
```

```
F1 2018-05-25 23:18:55,166 INFO Graph Rewriter optimizations enabled
Converted 275 variables to const ops.
F1 2018-05-25 23:19:03,867 INFO 2953 ops in the final graph.
```

Load the saved model for scoring

To use the saved model, load the model into memory with the 'load' function. You only need to load once.

```
my_detector_loaded = TFFasterRCNN.load(save_model_path)
```

After the model is loaded, it can be used to score an image or a list of images. For a single image, a dictionary is returned with keys such as 'detection_boxes', 'detection_scores', and 'num_detections'. If the input is a list of images, a list of dictionary is returned, with one dictionary corresponding to one image.

```
detections_dict = my_detector_loaded.score(image_path)
```

The detected objects with scores above 0.5, including labels, scores, and coordinates can be printed out.

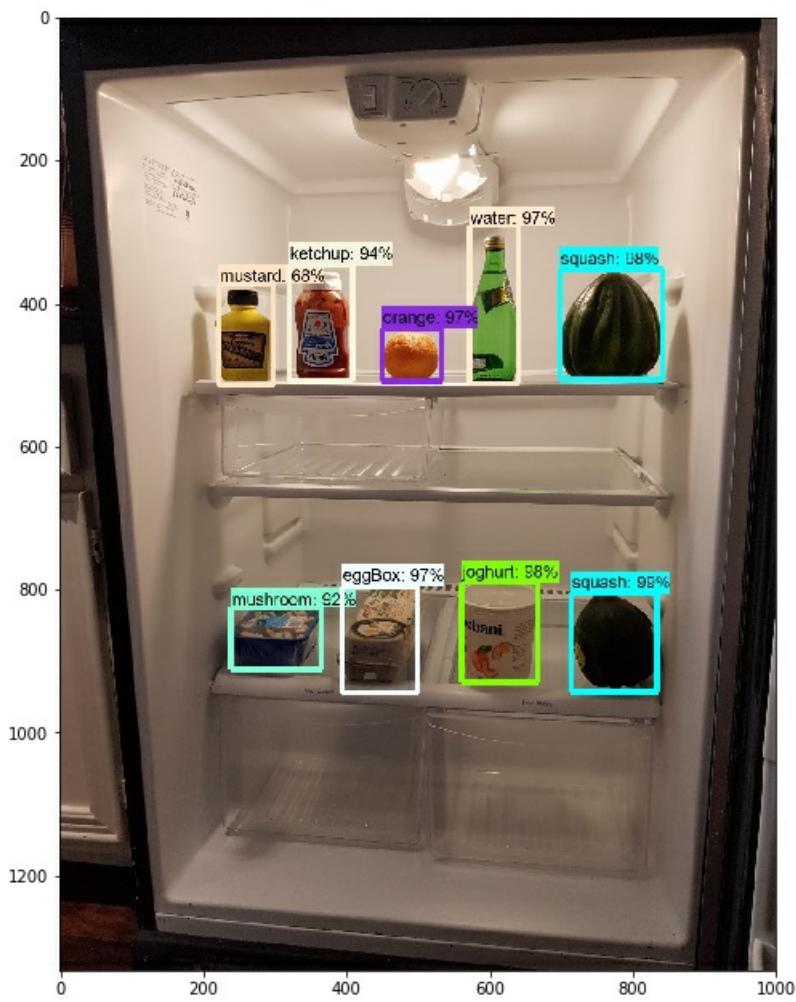
```
look_up = dict((v,k) for k,v in my_detector.class_map.items())
n_obj = 0
for i in range(detections_dict['num_detections']):
    if detections_dict['detection_scores'][i] > 0.5:
        n_obj += 1
        print("Object {}: label={:11}, score={:.2f}, location=(top: {:.2f}, left: {:.2f}, bottom: {:.2f},
right: {:.2f})".format(
            i, look_up[detections_dict['detection_classes'][i]],
            detections_dict['detection_scores'][i],
            detections_dict['detection_boxes'][i][0],
            detections_dict['detection_boxes'][i][1],
            detections_dict['detection_boxes'][i][2],
            detections_dict['detection_boxes'][i][3]))
print("\nFound {} objects in image {}".format(n_obj, image_path))
```

```
Object 0: label=squash      , score=0.99, location=(top: 0.74, left: 0.30, bottom: 0.84, right: 0.42)
Object 1: label=squash      , score=0.98, location=(top: 0.27, left: 0.21, bottom: 0.37, right: 0.33)
Object 2: label=orange      , score=0.98, location=(top: 0.31, left: 0.39, bottom: 0.37, right: 0.48)
Object 3: label=joghurt     , score=0.98, location=(top: 0.57, left: 0.29, bottom: 0.67, right: 0.39)
Object 4: label=eggBox       , score=0.97, location=(top: 0.41, left: 0.53, bottom: 0.49, right: 0.69)
Object 5: label=water        , score=0.95, location=(top: 0.23, left: 0.51, bottom: 0.37, right: 0.57)
Object 6: label=mustard      , score=0.88, location=(top: 0.61, left: 0.47, bottom: 0.66, right: 0.57)
Object 7: label=ketchup      , score=0.80, location=(top: 0.62, left: 0.62, bottom: 0.68, right: 0.72)
```

```
Found 8 objects in image ../sample_data/foods/test\JPEGImages\10.jpg.
```

Visualize the scores just like before.

```
path_save = "./scored_images/scored_image_frozen_graph.jpg"
ax = detection_utils.visualize(image_path, detections_dict, path_save=path_save, image_size=(8, 12))
# ax.get_figure() # use this code extract the returned image
```



Operationalization: deploy and consume

Operationalization is the process of publishing models and code as web services and the consumption of these services to produce business results.

Once your model is trained, you can deploy that model as a web service for consumption using [Azure Machine Learning CLI](#). Your models can be deployed to your local machine or Azure Container Service (ACS) cluster. Using ACS, you can scale your web service manually or use the autoscaling functionality.

Sign in with Azure CLI

Using an [Azure](#) account with a valid subscription, log in using the following CLI command:

```
az login
```

- To switch to another Azure subscription, use the command:

```
az account set --subscription [your subscription name]
```

- To see the current model management account, use the command:

```
az ml account modelmanagement show
```

Create and set your cluster deployment environment

You only need to set your deployment environment once. If you don't have one yet, set up your deployment environment now using [these instructions](#).

To see your active deployment environment, use the following CLI command:

```
az ml env show
```

Sample Azure CLI command to create and set deployment environment

```

az provider register -n Microsoft.MachineLearningCompute
az provider register -n Microsoft.ContainerRegistry
az provider register -n Microsoft.ContainerService
az ml env setup --cluster -n [your environment name] -l [Azure region e.g. westcentralus] [-g [resource group]]
az ml env set -n [environment name] -g [resource group]
az ml env cluster

```

Manage web services and deployments

The following APIs can be used to deploy models as web services, manage those web services, and manage deployments.

TASK	API
Create deployment object	<code>deploy_obj = AMLDeployment(deployment_name=deployment_name, associated_DNNModel=dnn_model, aml_env="cluster")</code>
Deploy web service	<code>deploy_obj.deploy()</code>
Score image	<code>deploy_obj.score_image(local_image_path_or_image_url)</code>
Delete web service	<code>deploy_obj.delete()</code>
Build docker image without web service	<code>deploy_obj.build_docker_image()</code>
List existing deployment	<code>AMLDisclosure.list_deployment()</code>
Delete if the service exists with the deployment name	<code>AMLDisclosure.delete_if_service_exist(deployment_name)</code>

API documentation: Consult the [package reference documentation](#) for the detailed reference for each module and class.

CLI reference: For more advanced operations related to deployment, refer to the [model management CLI reference](#).

Deployment management in Azure portal: You can track and manage your deployments in the [Azure portal](#). From the Azure portal, find your Machine Learning Model Management account page using its name. Then go to the Model Management account page > Model Management > Services.

```

# ##### OPTIONAL - Interactive CLI setup helper #####
# # Interactive CLI setup helper, including model management account and deployment environment.
# # If you haven't setup you CLI before or if you want to change you CLI settings, you can use this block to
# help you interactively.
# # UNCOMMENT THE FOLLOWING LINES IF YOU HAVE NOT CREATED OR SET THE MODEL MANAGEMENT ACCOUNT AND DEPLOYMENT
# ENVIRONMENT

# from azuremltkbase.deployment import CliSetup
# CliSetup().run()

```

```

from cvtk.operationalization import AMLDeployment

# set deployment name
deployment_name = "wsdeployment"

# Create deployment object
# It will use the current deployment environment (you can check it with CLI command "az ml env show").
deploy_obj = AMLDeployment(deployment_name=deployment_name, aml_env="cluster", associated_DNNModel=my_detector,
replicas=1)

# Alternatively, you can provide azure machine learning deployment cluster name (environment name) and resource
group name
# to deploy your model. It will use the provided cluster to deploy. To do that, please uncomment the following
lines to create
# the deployment object.

# azureml_rscgroup = "<resource group>"
# cluster_name = "<cluster name>"
# deploy_obj = AMLDeployment(deployment_name=deployment_name, associated_DNNModel=my_detector,
# #                                     aml_env="cluster", cluster_name=cluster_name, resource_group=azureml_rscgroup,
# replicas=1)

# Check if the deployment name exists, if yes remove it first.
if deploy_obj.is_existing_service():
    AMLDeployment.delete_if_service_exist(deployment_name)

# create the webservice
print("Deploying to Azure cluster...")
deploy_obj.deploy()
print("Deployment DONE")

```

Consume the web service

Once you created the webservice, you can score images with the deployed webservice. You have several options:

- You can directly score the webservice with the deployment object with:
`deploy_obj.score_image(image_path_or_url)`
- Or, you can use the Service endpoint url and Service key (None for local deployment) with:
`AMLDeployment.score_existing_service_with_image(image_path_or_url, service_endpoint_url,`
`service_key=None)`
- Form your http requests directly to score the webservice endpoint (For advanced users).

Score with existing deployment object

```
deploy_obj.score_image(image_path_or_url)
```

```

# Score with existing deployment object

# Score local image with file path
print("Score local image with file path")
image_path_or_url = data_train.images[0].storage_path
print("Image source:",image_path_or_url)
serialized_result_in_json = deploy_obj.score_image(image_path_or_url, image_resize_dims=[224,224])
print("serialized_result_in_json:", serialized_result_in_json[:50])

# Score image url and remove image resizing
print("Score image url")
image_path_or_url = "https://cvtkdata.blob.core.windows.net/publicimages/microsoft_logo.jpg"
print("Image source:",image_path_or_url)
serialized_result_in_json = deploy_obj.score_image(image_path_or_url)
print("serialized_result_in_json:", serialized_result_in_json[:50])

```

```
# Time image scoring
import timeit

num_images = 3
for img_index, img_obj in enumerate(data_train.images[:num_images]):
    print("Calling API for image {} of {}: {}".format(img_index, num_images, img_obj.name))
    tic = timeit.default_timer()
    return_json = deploy_obj.score_image(img_obj.storage_path, image_resize_dims=[224,224])
    print("    Time for API call: {:.2f} seconds".format(timeit.default_timer() - tic))
```

Score with service endpoint url and service key

```
AMLDeployment.score_existing_service_with_image(image_path_or_url, service_endpoint_url, service_key=None)
```

```
# Import related classes and functions
from cvtk.operationalization import AMLDeployment

service_endpoint_url = "http://xxx" # please replace with your own service url
service_key = "xxx" # please replace with your own service key

# score image url
image_path_or_url = "https://cvtkdata.blob.core.windows.net/publicimages/microsoft_logo.jpg"
print("Image source:",image_path_or_url)
serialized_result_in_json =
    AMLDeployment.score_existing_service_with_image(image_path_or_url,service_endpoint_url, service_key =
        service_key, image_resize_dims=[224,224])
print("serialized_result_in_json:", serialized_result_in_json[:50])
```

Score endpoint with http request directly

Following is some example code to form the http request directly in Python. You can do it in other programming languages.

```

def score_image_with_http(image, service_endpoint_url, service_key=None, parameters={}):
    """Score local image with http request

    Args:
        image (str): Image file path
        service_endpoint_url(str): web service endpoint url
        service_key(str): Service key. None for local deployment.
        parameters (dict): Additional request parameters in dictionary. Default is {}.

    Returns:
        str: serialized result
    """
    import requests
    from io import BytesIO
    import base64
    import json

    if service_key is None:
        headers = {'Content-Type': 'application/json'}
    else:
        headers = {'Content-Type': 'application/json',
                   "Authorization": ('Bearer ' + service_key)}
    payload = []
    encoded = None

    # Read image
    with open(image, 'rb') as f:
        image_buffer = BytesIO(f.read()) ## Getting an image file represented as a BytesIO object

    # Convert your image to base64 string
    # image_in_base64 : "b'{base64}'"
    encoded = base64.b64encode(image_buffer.getvalue())
    image_request = {"image_in_base64": "{0}".format(encoded), "parameters": parameters}
    payload.append(image_request)
    body = json.dumps(payload)
    r = requests.post(service_endpoint_url, data=body, headers=headers)
    try:
        result = json.loads(r.text)
        json.loads(result[0])
    except:
        raise ValueError("Incorrect output format. Result cant not be parsed: " + r.text)
    return result[0]

```

Parse serialized result from webservice

The result from the web service is in json string that can be parsed.

```

image_path_or_url = image_path
print("Image source:",image_path_or_url)
serialized_result_in_json = deploy_obj.score_image(image_path_or_url)
print("serialized_result_in_json:", serialized_result_in_json[:50])

```

```

# Parse result from json string
import numpy as np
parsed_result = TFFasterRCNN.parse_serialized_result(serialized_result_in_json)
print("Parsed result:", parsed_result)

```

```
ax = detection_utils.visualize(image_path, parsed_result)
path_save = "./scored_images/scored_image_web.jpg"
path_save_dir = os.path.dirname(os.path.abspath(path_save))
os.makedirs(path_save_dir, exist_ok=True)
ax.get_figure().savefig(path_save)
```

Next steps

Learn more about Azure Machine Learning Package for Computer Vision in these articles:

- Read the [package overview](#).
- Explore the [reference documentation](#) for this package.
- Learn about [other Python packages for Azure Machine Learning](#).

Build and deploy image similarity models with Azure Machine Learning

10/25/2018 • 2 minutes to read • [Edit Online](#)

NOTE

This article is deprecated. Support for earlier versions of this service will end incrementally. [View the support timeline](#). Start using the latest version with this [quickstart](#).

In this article, learn how to use Azure Machine Learning Package for Computer Vision (AMLPCV) to train, test, and deploy an image similarity model. For an overview of this package and its detailed reference documentation, see <https://aka.ms/aml-packages/vision>.

A large number of problems in the computer vision domain can be solved using image similarity. These problems include building models that answer questions such as:

- *Is an OBJECT present in the image? For example, "dog", "car", "ship", and so on*
- *What class of eye disease severity is evinced by this patient's retinal scan?*

When building and deploying this model with AMLPCV, you go through the following steps:

1. Dataset creation
2. Image visualization and annotation
3. Image augmentation
4. Deep Neural Network (DNN) model definition
5. Classifier training
6. Evaluation and visualization
7. Web service deployment
8. Web service load testing

CNTK is used as the deep learning framework, training is performed locally on a GPU powered machine such as the ([Deep learning Data Science VM](#)), and deployment uses the Azure ML Operationalization CLI.

Prerequisites

1. If you don't have an Azure subscription, create a [free account](#) before you begin.
2. The following accounts and application must be set up and installed:
 - An Azure Machine Learning Experimentation account
 - An Azure Machine Learning Model Management account
 - Azure Machine Learning Workbench installedIf these three are not yet created or installed, follow the [Azure Machine Learning Quickstart and Workbench installation](#) article.
3. The Azure Machine Learning Package for Computer Vision must be installed. Learn how to install this package as described at <https://aka.ms/aml-packages/vision>.

Sample data and notebook

Get the Jupyter notebook

Download the notebook to run the sample described here yourself.

Get the Jupyter notebook

Load the sample data

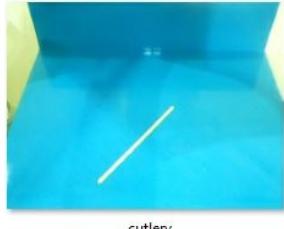
The following example uses a dataset consisting of 63 tableware images. Each image is labeled as belonging to one of four different classes (bowl, cup, cutlery, plate). The number of images in this example is small so that this sample can be executed quickly. In practice at least 100 images per class should be provided. All images are located at `"./sample_data/imgs_recycling/"`, in subdirectories called "bowl", "cup", "cutlery", and "plate".



bowl



cup



cutlery



plate

Next steps

Learn more about Azure Machine Learning Package for Computer Vision in these articles:

- Learn how to [improve the accuracy of this model](#).
- Read the [package overview](#).
- Explore the [reference documentation](#) for this package.
- Learn about [other Python packages for Azure Machine Learning](#).

Improve the accuracy of computer vision models

9/24/2018 • 6 minutes to read • [Edit Online](#)

NOTE

This article is deprecated. Support for earlier versions of this service will end incrementally. [View the support timeline](#). Start using the latest version with this [quickstart](#).

With the **Azure Machine Learning Package for Computer Vision**, you can build and deploy image classification, object detection, and image similarity models. Learn more about this package and how to install it.

In this article, you learn how to fine-tune these models to increase their accuracy.

Accuracy of image classification models

The Computer Vision Package is shown to give good results for a wide variety of datasets. However, as is true for most machine learning projects, getting the best possible results for a new dataset requires careful parameter tuning, as well as evaluating different design decisions. The following sections provide guidance on how to improve accuracy on a given dataset, that is, what parameters are most promising to optimize first, what values for these parameters one should try, and what pitfalls to avoid.

Generally speaking, training Deep Learning models comes with a trade-off between training time versus model accuracy. The Computer Vision Package has pre-set default parameters (see first row in the table below) which focus on fast training speed while typically producing high accuracy models. This accuracy can often be improved further using, for example, higher image resolution or deeper models, however at the cost of increasing training time by a factor of 10x or more.

It is recommended that you first work with the default parameters, train a model, inspect the results, correct ground truth annotations as needed, and only then try parameters that slow down training time (see table below suggested parameter values). An understanding of these parameters while technically not necessary is however recommended.

Best practices and tips

- Data quality: the training and test sets should be of high quality. That is, the images are annotated correctly, ambiguous images removed (for example where it is unclear to a human eye if the image shows a tennis ball or a lemon), and the attributes are mutually exclusive (that is, each image belongs to exactly one attribute).
- Before refining the DNN, an SVM classifier should be trained using a pre-trained and fixed DNN as featurizer. This is supported in Computer Vision Package and does not require long to train since the DNN itself is not modified. Even this simple approach often achieves good accuracies and hence represents a strong baseline. The next step is then to refine the DNN that should give better accuracy.
- If the object-of-interest is small in the image, then Image classification approaches are known to not work well. In such cases, consider using an object detection approach such as Computer Vision Package's Faster R-CNN based on Tensorflow.
- The more training data the better. As a rule-of-thumb, one should have at least 100 examples for each class, that is, 100 images for "dog", 100 images for "cat", etc. Training a model with fewer images is possible but might not produce good results.

- The training images need to reside locally on the machine with the GPU, and be on an SSD drive (not an HDD). Otherwise, latency from image reading can drastically reduce training speed (by even a factor of 100x).

Parameters to optimize

Finding optimal values for these parameters is important and can often improve accuracy significantly:

- Learning rate (`lr_per_mb`): The arguably most important parameter to get right. If the accuracy on the training set after DNN refinement is above ~5%, then most likely the learning rate is either too high, or the number of training epochs too low. Especially with small datasets, the DNN tends to over-fit on the training data, however in practice this will lead to good models on the test set. We typically use 15 epochs where the initial learning rate is reduced twice; training using more epochs can in some cases improve performance.
- Input resolution (`image_dims`): The default image resolution is 224x224 pixels. Using higher image resolution of, for example, 500x500 pixels or 1000x1000 pixels can significantly improve accuracy but slows down DNN refinement. The Computer Vision Package expects the input resolution to be a tuple of (color-channels, image-width, image-height), for example (3, 224, 224), where the number of color channels has to be set to 3 (the Red-Green-Blue bands).
- Model architecture(`base_model_name`): Try using deeper DNNs such as ResNet-34 or ResNet-50 instead of the default ResNet-18 model. The Resnet-50 model is not only deeper, but its output of the penultimate layer is of size 2048 floats (vs. 512 floats of the ResNet-18 and ResNet-34 models). This increased dimensionality can be especially beneficial when keeping the DNN fixed and instead training an SVM classifier.
- Minibatch size (`mb_size`): High minibatch sizes will lead to faster training time however at the expense of an increased DNN memory consumption. Hence, when selecting deeper models (for example, ResNet-50 versus ResNet-18) and/or higher image resolution (500*500 pixels versus 224*224 pixels), one typically has to reduce the minibatch size to avoid out-of-memory errors. When changing the minibatch size, often also the learning rate needs to be adjusted as can be seen in the table below.
- Drop-out rate (`dropout_rate`) and L2-regularizer (`l2_reg_weight`): DNN over-fitting can be reduced by using a dropout rate of 0.5 (default is 0.5 in Computer Vision Package) or more, and by increasing the regularizer weight (default is 0.0005 in Computer Vision Package). Note though that especially with small datasets DNN over-fitting is hard and often impossible to avoid.

Parameter definitions

- **Learning rate:** step size used during gradient descent learning. If set too low then the model will take many epochs to train, if set too high then the model will not converge to a good solution. Typically a schedule is used where the learning rate is reduced after a certain number of epochs. E.For example the learning rate schedule `[0.05]*7 + [0.005]*7 + [0.0005]` corresponds to using an initial learning rate of 0.05 for the first seven epochs, followed by a 10x reduced learning rate of 0.005 for another seven epochs, and finally fine-tuning the model for a single epoch with a 100x reduced learning rate of 0.0005.
- **Minibatch size:** GPUs can process multiple images in parallel to speed up computation. These parallel processed images are also referred as a minibatch. The higher the minibatch size the faster training will be, however at the expense of an increased DNN memory consumption.

Recommended parameter values

The table below provides different parameter sets that were shown to produce high accuracy models on a wide variety of image classification tasks. The optimal parameters depend on the specific dataset and on the exact GPU used, hence the table should be seen as a guideline only. After trying these parameters, consider also image resolutions of more than 500x500 pixels, or deeper models such as Resnet-101 or Resnet-152.

The first row in the table corresponds to the default parameters that are set inside Computer Vision Package. All

other rows take longer to train (indicated in the first column) however at the benefit of increased accuracy (see the second column for the average accuracy over three internal datasets). For example, the parameters in the last row take 5-15x longer to train, however resulted in increased (averaged) accuracy on three internal test sets from 82.6% to 92.8%.

Deeper models and higher input resolution take up more DNN memory, and hence the minibatch size needs to be reduced with increased model complexity to avoid out-of-memory-errors. As can be seen in the table below, it is beneficial to decrease the learning rate by a factor of two whenever decreasing the minibatch size by the same multiplier. The minibatch size might need to get reduced further on GPUs with smaller amounts of memory.

TRAINING TIME (ROUGH ESTIMATE)	EXAMPLE ACCURACY	MINIBATCH SIZE (MB_SIZE)	LEARNING RATE (LR_PER_MB)	IMAGE RESOLUTION (IMAGE_DIMS)	DNN ARCHITECTURE (BASE_MODEL_NA ME)
1x (reference)	82.6%	32	[0.05]*7 + [0.005]*7 + [0.0005]	(3, 224, 224)	ResNet18_Image Net_CNTK
2-5x	90.2%	16	[0.025]*7 + [0.0025]*7 + [0.00025]	(3, 500, 500)	ResNet18_Image Net_CNTK
2-5x	87.5%	16	[0.025]*7 + [0.0025]*7 + [0.00025]	(3, 224, 224)	ResNet50_Image Net_CNTK
5-15x	92.8%	8	[0.01]*7 + [0.001]*7 + [0.0001]	(3, 500, 500)	ResNet50_Image Net_CNTK

Next steps

For information about the Azure Machine Learning Package for Computer Vision:

- Check out the reference documentation
- Learn about [other Python packages for Azure Machine Learning](#)

Build and deploy forecasting models with Azure Machine Learning

11/29/2018 • 25 minutes to read • [Edit Online](#)

NOTE

This article is deprecated. Support for earlier versions of this service will end incrementally. [View the support timeline](#). Start using the latest version with this [quickstart](#).

In this article, learn how to use **Azure Machine Learning Package for Forecasting** (AMLPF) to quickly build and deploy a forecasting model. The workflow is as follows:

1. Load and explore data
2. Create features
3. Train and select the best model
4. Deploy the model and consume the web service

Consult the [package reference documentation](#) for the full list of transformers and models as well as the detailed reference for each module and class.

Prerequisites

1. If you don't have an Azure subscription, create a [free account](#) before you begin.
2. The following accounts and application must be set up and installed:
 - An Azure Machine Learning Experimentation account
 - An Azure Machine Learning Model Management account
 - Azure Machine Learning Workbench installedIf these three are not yet created or installed, follow the [Azure Machine Learning Quickstart and Workbench installation](#) article.
3. The Azure Machine Learning Package for Forecasting must be installed. Learn how to [install this package here](#).

Sample data and Jupyter notebook

Sample workflow

The example follows the workflow:

1. **Ingest Data:** Load the dataset and convert it into TimeSeriesDataFrame. This dataframe is a time series data structure provided by Azure Machine Learning Package for Forecasting, herein referred to as **AMLPF**.
2. **Create Features:** Use various featurization transformers provided by AMLPF to create features.
3. **Train and Select Best Model:** Compare the performance of various univariate time series models and machine learning models.
4. **Deploy Model:** Deploy the trained model pipeline as a web service via Azure Machine Learning Workbench so it can be consumed by others.

Get the Jupyter notebook

Download the notebook to run the code samples described herein yourself.

Get the Jupyter notebook

Explore the sample data

The machine learning forecasting examples in the follow code samples rely on the [University of Chicago's Dominick's Finer Foods dataset](#) to forecast orange juice sales. Dominick's was a grocery chain in the Chicago metropolitan area.

Import any dependencies for this sample

These dependencies must be imported for the following code samples:

```
import pandas as pd
import numpy as np
import math
import pkg_resources
from datetime import timedelta
import matplotlib
matplotlib.use('agg')
%matplotlib inline
from matplotlib import pyplot as plt

from sklearn.linear_model import Lasso, ElasticNet
from sklearn.ensemble import RandomForestRegressor, GradientBoostingRegressor
from sklearn.neighbors import KNeighborsRegressor

from ftk import TimeSeriesDataFrame, ForecastDataFrame, AzureMLForecastPipeline
from ftk.ts_utils import last_n_periods_split

from ftk.transforms import TimeSeriesImputer, TimeIndexFeaturizer, DropColumns
from ftk.transforms.grain_index_featurizer import GrainIndexFeaturizer
from ftk.models import Arima, SeasonalNaive, Naive, RegressionForecaster, ETS, BestOfForecaster
from ftk.models.forecaster_union import ForecasterUnion
from ftk.model_selection import TSGridSearchCV, RollingOriginValidator

from azuremltkbase.deployment import AMLSettings
from ftk.operationalization.forecast_webservice_factory import ForecastWebserviceFactory
from ftk.operationalization import ScoreContext

from ftk.data import get_a_year_of_daily_weather_data
print('imports done')
```

```
imports done
```

Load data and explore

This code snippet shows the typical process of starting with a raw data set, in this case the [data from Dominick's Finer Foods](#). You can also use the convenience function [load_dominicks_oj_data](#).

```
# Load the data into a pandas DataFrame
csv_path = pkg_resources.resource_filename('ftk', 'data/dominicks_oj/dominicks_oj.csv')
whole_df = pd.read_csv(csv_path, low_memory = False)
whole_df.head()
```

	ST OR E	BR AN D	W EE K	LO G M OV E	FE AT	PR ICE	AG E6 0	ED UC	ET HN IC	IN CO ME	HH LA RG E	W OR K W O M	HV AL 15 0	SS TR DI ST	SS TR VO L	CP DI ST 5	CP W VO L5
0	2	tropicana	40	9.02	0	3.87	0.23	0.25	0.11	10.55	0.10	0.30	0.46	2.11	1.14	1.93	0.38
1	2	tropicana	46	8.72	0	3.87	0.23	0.25	0.11	10.55	0.10	0.30	0.46	2.11	1.14	1.93	0.38
2	2	tropicana	47	8.25	0	3.87	0.23	0.25	0.11	10.55	0.10	0.30	0.46	2.11	1.14	1.93	0.38
3	2	tropicana	48	8.99	0	3.87	0.23	0.25	0.11	10.55	0.10	0.30	0.46	2.11	1.14	1.93	0.38
4	2	tropicana	50	9.09	0	3.87	0.23	0.25	0.11	10.55	0.10	0.30	0.46	2.11	1.14	1.93	0.38

The data consist of weekly sales by brand and store. The logarithm of the quantity sold is in the *logmove* column.

The data also includes some customer demographic features.

To model the time series, you need to extract the following elements from this dataframe:

- A date/time axis
- The sales quantity to be forecast

```
# The sales are contained in the 'logmove' column.
# Values are logarithmic, so exponentiate and round them to get quantity sold
def expround(x):
    return math.floor(math.exp(x) + 0.5)
whole_df['Quantity'] = whole_df['logmove'].apply(expround)

# The time axis is in the 'week' column
# This is the week offset from the week of 1989-09-07 through 1989-09-13 inclusive
# Create new datetime columns containing the start and end of each week period
weekZeroStart = pd.to_datetime('1989-09-07 00:00:00')
weekZeroEnd = pd.to_datetime('1989-09-13 23:59:59')
whole_df['WeekFirstDay'] = whole_df['week'].apply(lambda n: weekZeroStart + timedelta(weeks=n))
whole_df['WeekLastDay'] = whole_df['week'].apply(lambda n: weekZeroEnd + timedelta(weeks=n))
whole_df[['store','brand','WeekLastDay','Quantity']].head()
```

	STORE		BRAND		WEEKLASTDAY		QUANTITY	
0	2		tropicana		1990-06-20 23:59:59		8256	

	STORE	BRAND	WEEKLASTDAY	QUANTITY
1	2	tropicana	1990-08-01 23:59:59	6144
2	2	tropicana	1990-08-08 23:59:59	3840
3	2	tropicana	1990-08-15 23:59:59	8000
4	2	tropicana	1990-08-29 23:59:59	8896

```
nseries = whole_df.groupby(['store', 'brand']).ngroups
print('{} time series in the data frame.'.format(nseries))
```

249 time series in the data frame.

The data contains approximately 250 different combinations of store and brand in a data frame. Each combination defines its own time series of sales.

You can use the [TimeSeriesDataFrame](#) class to conveniently model multiple series in a single data structure using the *grain*. The grain is specified by the `store` and `brand` columns.

The difference between *grain* and *group* is that grain is always physically meaningful in the real world, while group doesn't have to be. Internal package functions use group to build a single model from multiple time series if the user believes this grouping helps improve model performance. By default, group is set to be equal to grain, and a single model is built for each grain.

```
# Create a TimeSeriesDataFrame
# Use end of period as the time index
# Store and brand combinations label the grain
# i.e. there is one time series for each unique pair of store and grain
whole_tsdf = TimeSeriesDataFrame(whole_df,
                                 grain_colnames=['store', 'brand'],
                                 time_colname='WeekLastDay',
                                 ts_value_colname='Quantity',
                                 group_colnames='store')

whole_tsdf[['Quantity']].head()
```

			QUANTITY
WEEKLASTDAY	STORE	BRAND	
1990-06-20 23:59:59	2	TROPICANA	8256
1990-08-01 23:59:59	2	TROPICANA	6144
1990-08-08 23:59:59	2	TROPICANA	3840
1990-08-15 23:59:59	2	TROPICANA	8000
1990-08-29 23:59:59	2	TROPICANA	8896

In the TimeSeriesDataFrame representation, the time axis and grain are now part of the data frame index, and

allow easy access to pandas datetime slicing functionality.

```
# sort so we can slice
whole_tsdf.sort_index(inplace=True)

# Get sales of dominick's brand orange juice from store 2 during summer 1990
whole_tsdf.loc[pd.IndexSlice['1990-06':'1990-09', 2, 'dominicks'], ['Quantity']]
```

WEEKLASTDAY	STORE	BRAND	QUANTITY
1990-06-20 23:59:59	2	DOMINICKS	10560
1990-08-01 23:59:59	2	DOMINICKS	8000
1990-08-08 23:59:59	2	DOMINICKS	6848
1990-08-15 23:59:59	2	DOMINICKS	2880
1990-08-29 23:59:59	2	DOMINICKS	1600
1990-09-05 23:59:59	2	DOMINICKS	25344
1990-09-12 23:59:59	2	DOMINICKS	10752
1990-09-19 23:59:59	2	DOMINICKS	6656
1990-09-26 23:59:59	2	DOMINICKS	6592

The [TimeSeriesDataFrame.ts_report](#) function generates a comprehensive report of the time series data frame. The report includes both a general data description as well as statistics specific to time series data.

```
whole_tsdf.ts_report()
```

```
----- Data Overview -----
<class 'ftk.time_series_data_frame.TimeSeriesDataFrame'>
MultiIndex: 28947 entries, (1990-06-20 23:59:59, 2, dominicks) to (1992-10-07 23:59:59, 137, tropicana)
Data columns (total 17 columns):
week          28947 non-null int64
logmove       28947 non-null float64
feat          28947 non-null int64
price         28947 non-null float64
AGE60         28947 non-null float64
EDUC          28947 non-null float64
ETHNIC        28947 non-null float64
INCOME         28947 non-null float64
HHLARGE        28947 non-null float64
WORKWOM        28947 non-null float64
HVAL150        28947 non-null float64
SSTRDIST       28947 non-null float64
SSTRVOL        28947 non-null float64
CPDISTS5      28947 non-null float64
CPWVOLS5      28947 non-null float64
Quantity       28947 non-null int64
WeekFirstDay   28947 non-null datetime64[ns]
dtypes: datetime64[ns](1), float64(13), int64(3)
```

```

memory usage: 3.8+ MB
----- Numerical Variable Summary -----
      week logmove    feat   price  AGE60   EDUC ETHNIC INCOME \
count 28947.00 28947.00 28947.00 28947.00 28947.00 28947.00 28947.00 28947.00
mean   100.46    9.17   0.24   2.28   0.17   0.23   0.16   10.62
std    34.69    1.02   0.43   0.65   0.06   0.11   0.19   0.28
min    40.00    4.16   0.00   0.52   0.06   0.05   0.02   9.87
25%    70.00    8.49   0.00   1.79   0.12   0.15   0.04  10.46
50%   101.00   9.03   0.00   2.17   0.17   0.23   0.07  10.64
75%   130.00   9.76   0.00   2.73   0.21   0.28   0.19  10.80
max   160.00  13.48   1.00   3.87   0.31   0.53   1.00  11.24

      HHLARGE WORKWOM HVAL150 SSTRDIST SSTRVOL CPDIST5 CPWVOL5 \
count 28947.00 28947.00 28947.00 28947.00 28947.00 28947.00 28947.00 28947.00
mean   0.12    0.36   0.34   5.10   1.21   2.12   0.44
std    0.03    0.05   0.24   3.47   0.53   0.73   0.22
min    0.01    0.24   0.00   0.13   0.40   0.77   0.09
25%    0.10    0.31   0.12   2.77   0.73   1.63   0.27
50%    0.11    0.36   0.35   4.65   1.12   1.96   0.38
75%    0.14    0.40   0.53   6.65   1.54   2.53   0.56
max   0.22    0.47   0.92  17.86   2.57   4.11   1.14

      Quantity
count 28947.00
mean 17312.21
std 27477.66
min 64.00
25% 4864.00
50% 8384.00
75% 17408.00
max 716416.00
----- Non-Numerical Variable Summary -----
      WeekFirstDay
count          28947
unique         121
top 1992-03-12 00:00:00
freq          249
first 1990-06-14 00:00:00
last 1992-10-01 00:00:00
----- Time Series Summary -----
Number of time series 249
Minimum time 1990-06-20 23:59:59
Maximum time 1992-10-07 23:59:59

Inferred frequencies
Number of ['store', 'brand']s with frequency W-WED 249
Use get_frequency_dict() method to explore ['store', 'brand']s with unusual frequency and clean data

Detected seasonalities
Number of ['store', 'brand']s with seasonality 1 190
Number of ['store', 'brand']s with seasonality 15 15
Number of ['store', 'brand']s with seasonality 14 11
Number of ['store', 'brand']s with seasonality 7 9
Number of ['store', 'brand']s with seasonality 6 8
Number of ['store', 'brand']s with seasonality 8 5
Number of ['store', 'brand']s with seasonality 2 4
Number of ['store', 'brand']s with seasonality 23 2
Number of ['store', 'brand']s with seasonality 3 1
Number of ['store', 'brand']s with seasonality 11 1
Number of ['store', 'brand']s with seasonality 12 1
Number of ['store', 'brand']s with seasonality 13 1
Number of ['store', 'brand']s with seasonality 47 1
Use get_seasonality_dict() method to explore ['store', 'brand']s with unusual seasonality and clean data
----- Value Column Summary -----
Value column      Quantity
Percentage of missing values 0.00
Percentage of zero values 0.00
Mean coefficient of variation 31688.52
Median coefficient of variation 24000.20

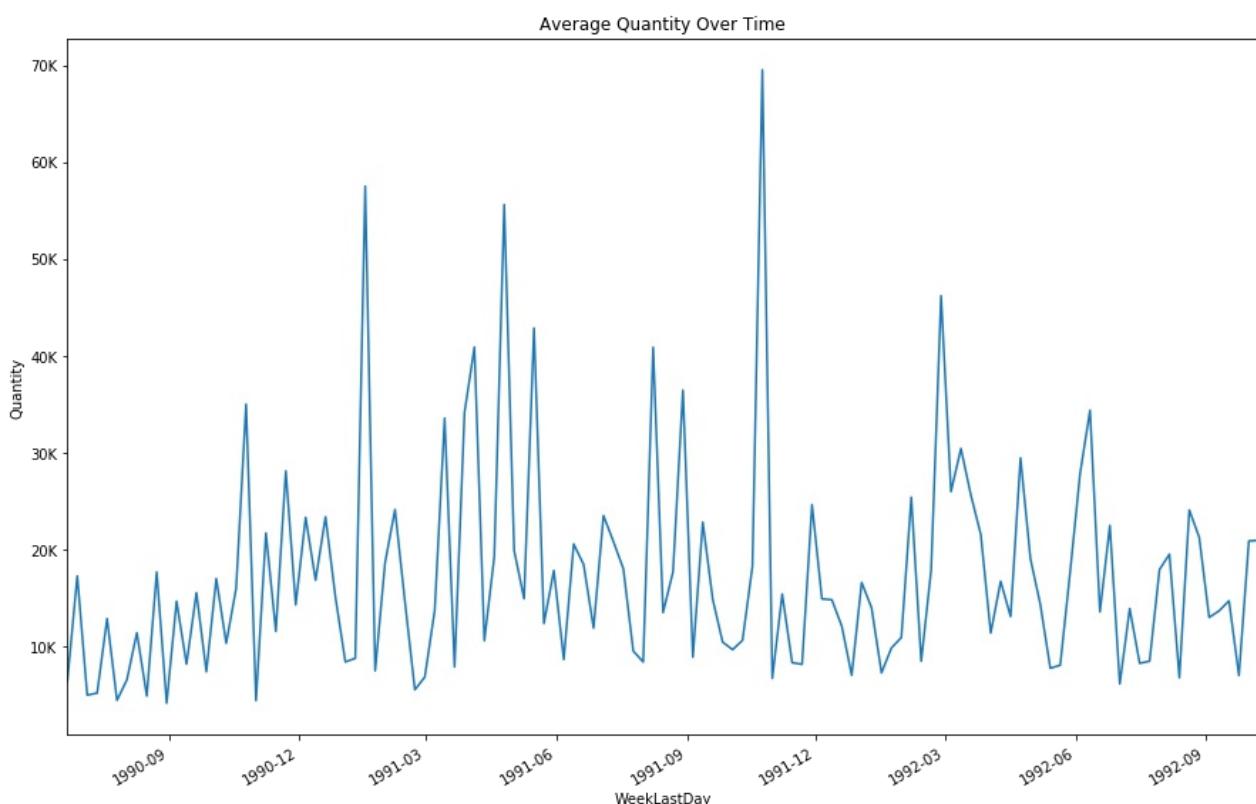
```

```
Minimum coefficient of variation      ['store', 'brand'] (48, 'tropicana'): 4475.53
Maximum coefficient of variation     ['store', 'brand'] (111, 'dominicks'): 193333.55
```

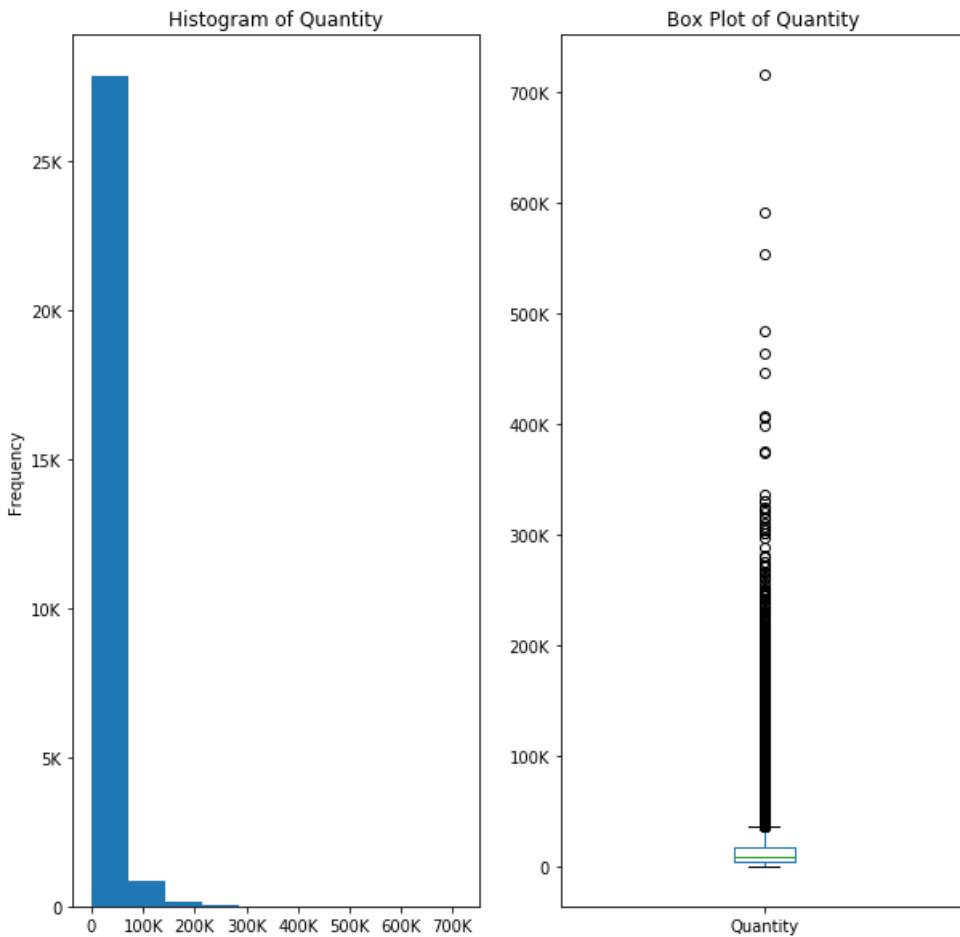
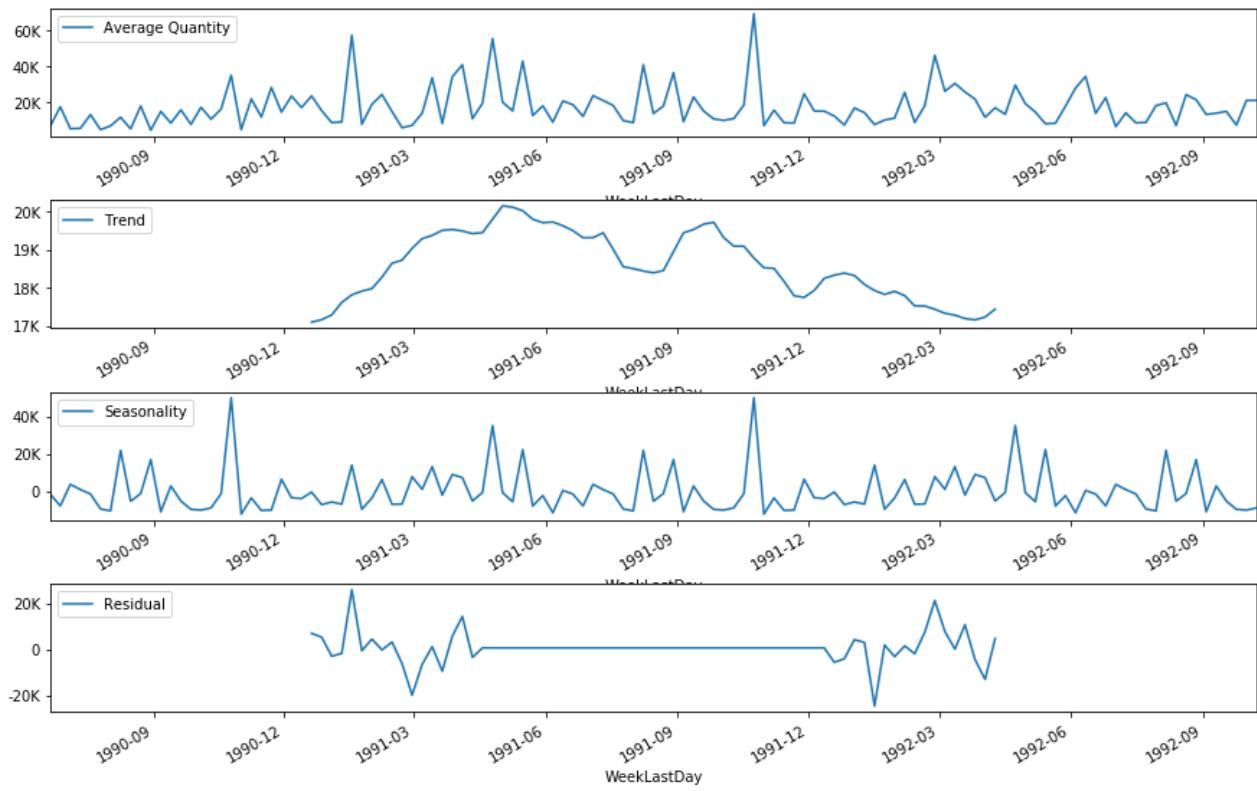
Correlation Matrix											
week	logmove	feat	price	AGE60	EDUC	ETHNIC	INCOME	HHLARGE	WORKWOM	\	
0	1.00	0.10	0.04	-0.21	-0.01	0.01	0.00	0.00	0.01	-0.00	
1	0.10	1.00	0.54	-0.43	0.09	0.00	0.06	-0.04	-0.06	-0.08	
2	0.04	0.54	1.00	-0.29	-0.00	0.00	0.00	-0.00	-0.00	0.00	
3	-0.21	-0.43	-0.29	1.00	0.04	0.02	0.04	-0.03	-0.04	-0.02	
4	-0.01	0.09	-0.00	0.04	1.00	-0.31	-0.09	-0.15	-0.32	-0.63	
5	0.01	0.00	0.00	0.02	-0.31	1.00	-0.34	0.66	-0.39	0.56	
6	0.00	0.06	0.00	0.04	-0.09	-0.34	1.00	-0.72	0.25	-0.29	
7	0.00	-0.04	-0.00	-0.03	-0.15	0.66	-0.72	1.00	-0.08	0.40	
8	0.01	-0.06	-0.00	-0.04	-0.32	-0.39	0.25	-0.08	1.00	-0.28	
9	-0.00	-0.08	0.00	-0.02	-0.63	0.56	-0.29	0.40	-0.28	1.00	
10	0.01	0.02	0.00	0.04	-0.11	0.89	-0.42	0.64	-0.48	0.45	
11	0.01	-0.00	0.00	0.08	0.07	-0.12	0.54	-0.41	0.06	-0.19	
12	-0.01	-0.09	-0.00	0.03	-0.05	-0.13	0.23	-0.26	0.04	-0.06	
13	-0.01	0.02	-0.00	-0.06	0.09	-0.20	-0.22	0.21	0.19	-0.13	
14	-0.00	-0.12	-0.00	-0.01	-0.09	0.28	-0.38	0.36	-0.20	0.37	
15	0.03	0.76	0.47	-0.36	0.08	-0.04	0.11	-0.08	-0.00	-0.10	

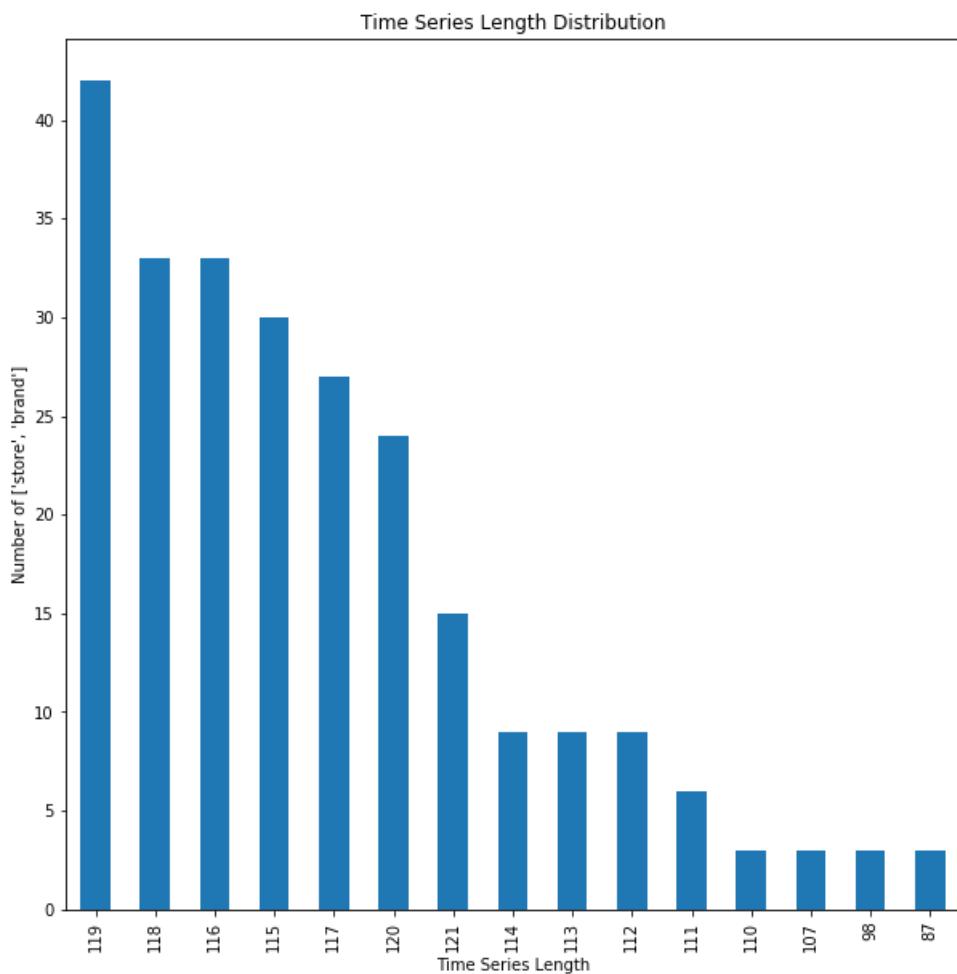
	HVAL150	SSTRDIST	SSTRVOL	CPDIST5	CPWVOL5	Quantity
0	0.01	0.01	-0.01	-0.01	-0.00	0.03
1	0.02	-0.00	-0.09	0.02	-0.12	0.76
2	0.00	0.00	-0.00	-0.00	-0.00	0.47
3	0.04	0.08	0.03	-0.06	-0.01	-0.36
4	-0.11	0.07	-0.05	0.09	-0.09	0.08
5	0.89	-0.12	-0.13	-0.20	0.28	-0.04
6	-0.42	0.54	0.23	-0.22	-0.38	0.11
7	0.64	-0.41	-0.26	0.21	0.36	-0.08
8	-0.48	0.06	0.04	0.19	-0.20	-0.00
9	0.45	-0.19	-0.06	-0.13	0.37	-0.10
10	1.00	-0.17	-0.24	-0.22	0.27	-0.04
11	-0.17	1.00	0.17	-0.11	-0.40	0.06
12	-0.24	0.17	1.00	-0.05	0.36	-0.02
13	-0.22	-0.11	-0.05	1.00	0.02	-0.00
14	0.27	-0.40	0.36	0.02	1.00	-0.11
15	-0.04	0.06	-0.02	-0.00	-0.11	1.00

You may call `TimeSeriesDataFrame.plot_scatter_matrix()` to get a correlation matrix plot. However, this is not recommended if your data is big.

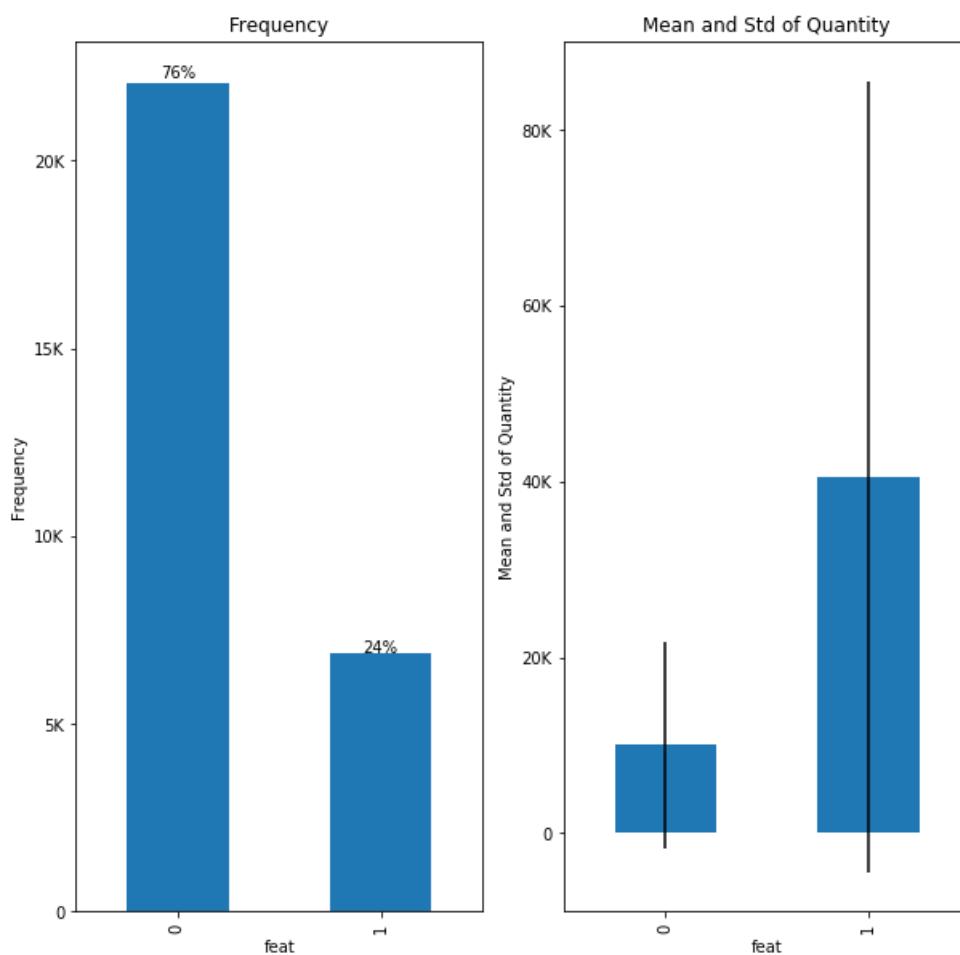


STL Decomposition

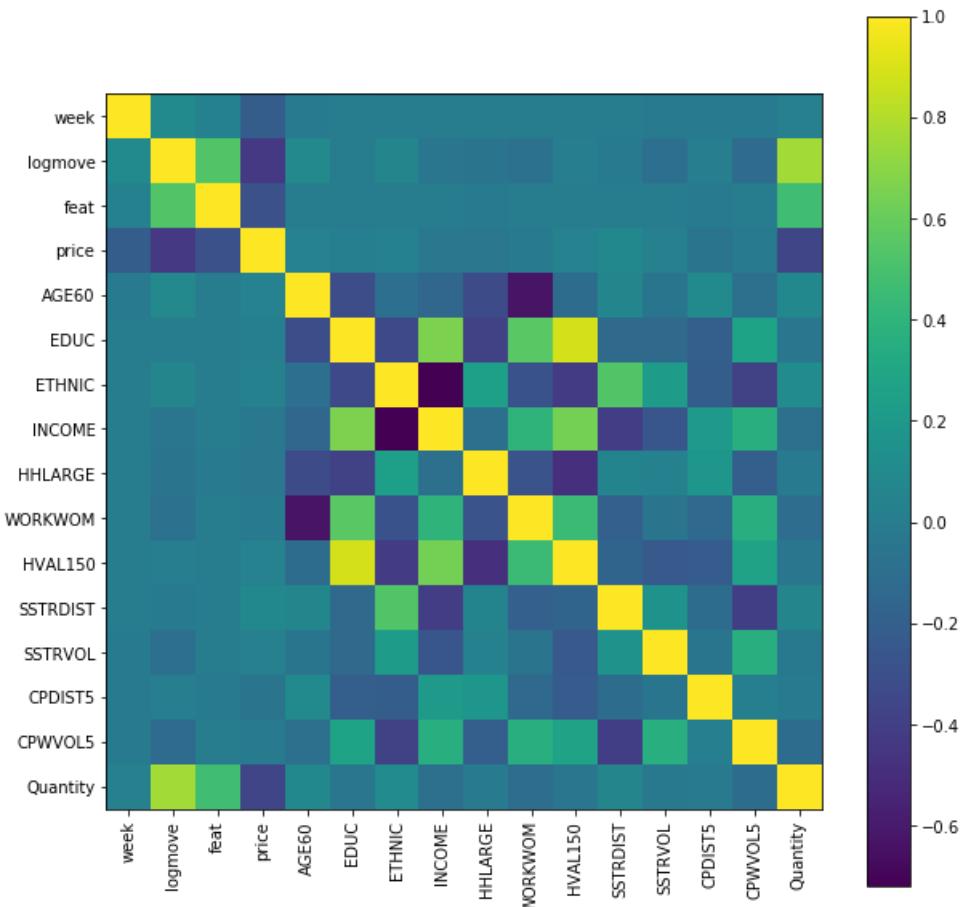




Inferred Categorical Column: **feat**



Numerical Variable Correlation Matrix Heat Map



Integrate with external data

Sometimes it's useful to integrate external data as additional features for forecasting. In this code sample, you join TimeSeriesDataFrame with external data related to weather.

```

# Load weather data
weather_1990 = get_a_year_of_daily_weather_data(year=1990)
weather_1991 = get_a_year_of_daily_weather_data(year=1991)
weather_1992 = get_a_year_of_daily_weather_data(year=1992)

# Preprocess weather data
weather_all = pd.concat([weather_1990, weather_1991, weather_1992])
weather_all.reset_index(inplace=True)

# Only use a subset of columns
weather_all = weather_all[['TEMP', 'DEWP', 'WDSP', 'PRCP']]

# Compute the WeekLastDay column, in order to merge with sales data
weather_all['WeekLastDay'] = pd.Series(
    weather_all.time_index - weekZeroStart,
    index=weather_all.time_index).apply(lambda n: weekZeroEnd + timedelta(weeks=math.floor(n.days/7)))

# Resample daily weather data to weekly data
weather_all = weather_all.groupby('WeekLastDay').mean()

# Set WeekLastDay as new time index
weather_all = TimeSeriesDataFrame(weather_all, time_colname='WeekLastDay')

# Merge weather data with sales data
whole_tsdf = whole_tsdf.merge(weather_all, how='left', on='WeekLastDay')
whole_tsdf.head()

```

			WEEK	LOGMOVE	FEATURE	PRICE	AGE	EDUC	ETHNIC	INCOME	HHLARGE	WORKWOM	..	SSTDIST	SSTRVOL	CPT5	CPWVL5	QUANTITY	WEEKFIRSTDAY	TEMP	DEWP	WDSP	PRCP	
WEEKDAY	STORE	BRAND																						
1990-06-23:59:59	2	DOMINICKS	40.	9.26	1.59	1.23	0.25	0.11	0.51	1.05	0.10	0.30	..	2.11	1.14	1.93	0.38	1.0560	1990-06-21:59:59	7.00	6.87	9.4	0.19	

WEEK	LOGMOVE	FEEAT	PRICE	AGE	EDUC	ETHNIC	INCOME	HLLARGE	WORKWOM	..	STRDIST	STRVOL	CPIST5	CPIST5	QUANTITY	WEEKFIRSTDAY	TEMP	DEWP	WDSP	PRCP
WEEKLASTDAY	STORE	BRAND																		
	MINUTE	40.	8.35	0.99	2.12	0.22	0.05	1.09	0.19	0.01	3.80	0.68	1.60	0.74	4.24	1.90	7.00	6.87	9.49	0.19
	MAID															-0.61	-1.14			

Preprocess data and impute missing values

Start by splitting the data into training set and a testing set with the `last_n_periods_split` utility function. The resulting testing set contains the last 40 observations of each time series.

```
train_tsdf, test_tsdf = last_n_periods_split(whole_tsdf, 40)
```

Basic time series models require contiguous time series. Check to see if the series are regular, meaning that they have a time index sampled at regular intervals, using the `check_regularity_by_grain` function.

```
ts_regularity = train_tsdf.check_regularity_by_grain()
print(ts_regularity[ts_regularity['regular'] == False])
```

[213 rows x 2 columns]

You can see that most of the series (213 out of 249) are irregular. An [imputation transform](#) is required to fill in missing sales quantity values. While there are many imputation options, the following sample code uses a linear interpolation.

```
# Use a TimeSeriesImputer to linearly interpolate missing values
imputer = TimeSeriesImputer(input_column='Quantity',
                             option='interpolate',
                             method='linear',
                             freq='W-WED')

train_imputed_tsdf = imputer.transform(train_tsdf)
```

After the imputation code is run, the series all have a regular frequency:

```
ts_regularity_imputed = train_imputed_tsdf.check_regularity_by_grain()
print(ts_regularity_imputed[ts_regularity_imputed['regular']] == False)
```

```
Empty DataFrame
Columns: [problems, regular]
Index: []
```

Univariate time series models

Now that you have cleaned up the data, you can begin modeling. Start by creating three univariate models: the "naive" model, the "seasonal naive" model, and an "ARIMA" model.

- The Naive forecasting algorithm uses the actual target variable value of the last period as the forecasted value of the current period.
- The Seasonal Naive algorithm uses the actual target variable value of the same time point of the previous season as the forecasted value of the current time point. Some examples include using the actual value of the same month of last year to forecast months of the current year; use the same hour of yesterday to forecast hours today.
- The exponential smoothing (ETS) algorithm generates forecasts by computing the weighted averages of past observations, with the weights decaying exponentially as the observations get older.
- The AutoRegressive Integrated Moving Average (ARIMA) algorithm captures the autocorrelation in time series data. For more information about ARIMA, see [this link](#)

Start by setting certain model parameters based on your data exploration.

```
oj_series_freq = 'W-WED'
oj_series_seasonality = 52
```

Initialize Models

```

# Initialize naive model.
naive_model = Naive(freq=oj_series_freq)

# Initialize seasonal naive model.
seasonal_naive_model = SeasonalNaive(freq=oj_series_freq,
                                       seasonality=oj_series_seasonality)

# Initialize ETS model.
ets_model = ETS(freq=oj_series_freq, seasonality=oj_series_seasonality)

# Initialize ARIMA(p,d,q) model.
arima_order = [2, 1, 0]
arima_model = Arima(oj_series_freq, arima_order)

```

Combine Multiple Models

The `ForecasterUnion` estimator allows you to combine multiple estimators and fit/predict on them using one line of code.

```

forecaster_union = ForecasterUnion(
    forecaster_list=[('naive', naive_model), ('seasonal_naive', seasonal_naive_model),
                     ('ets', ets_model), ('arima', arima_model)])

```

Fit and Predict

The estimators in AMLPF follow the same API as scikit-learn estimators: a fit method for model training and a predict method for generating forecasts.

Train models

Since these models are all univariate models, one model is fit to each grain of the data. Using AMLPF, all 249 models can be fit with just one function call.

```
forecaster_union_fitted = forecaster_union.fit(train_imputed_tsdf)
```

Forecast sales on test data

Similar to the fit method, you can create predictions for all 249 series in the testing data set with one call to the `predict` function.

```
forecaster_union_prediction = forecaster_union_fitted.predict(test_tsdf, retain_feature_column=True)
```

Evaluate model performance

Now you can calculate the forecast errors on the test set. You can use the mean absolute percentage error (MAPE) here. MAPE is the mean absolute percent error relative to the actual sales values. The `calc_error` function provides a few built-in functions for commonly used error metrics. You can also define our custom error function to calculate MedianAPE and pass it to the `err_fun` argument.

```

def calc_median_ape(y_true, y_pred):
    y_true = np.array(y_true).astype(float)
    y_pred = np.array(y_pred).astype(float)
    y_true_rm_na = y_true[~(np.isnan(y_true) | np.isnan(y_pred))]
    y_pred_rm_na = y_pred[~(np.isnan(y_true) | np.isnan(y_pred))]
    y_true = y_true_rm_na
    y_pred = y_pred_rm_na
    if len(y_true) == 0:
        # if there is no entries left after removing na data, return np.nan
        return(np.nan)
    y_true_rm_zero = y_true[y_true != 0]
    y_pred_rm_zero = y_pred[y_true != 0]
    if len(y_true_rm_zero) == 0:
        # if all values are zero, np.nan will be returned.
        return(np.nan)
    ape = np.abs((y_true_rm_zero - y_pred_rm_zero) / y_true_rm_zero) * 100
    median_ape = np.median(ape)
    return median_ape

```

```

forecaster_union_MAPE = forecaster_union_prediction.calc_error(err_name='MAPE',
                                                               by='ModelName')
forecaster_union_MedianAPE = forecaster_union_prediction.calc_error(err_name='MedianAPE',
                                                                     err_fun=calc_median_ape,
                                                                     by='ModelName')

univariate_model_errors = forecaster_union_MAPE.merge(forecaster_union_MedianAPE, on='ModelName')
univariate_model_errors

```

	MODELNAME	MAPE	MEDIANAPE
0	arima	126.57	66.49
1	ets	187.89	75.73
2	naive	103.57	59.14
3	seasonal_naive	180.54	65.99

Build machine learning models

In addition to traditional univariate models, Azure Machine Learning Package for Forecasting also enables you to create machine learning models.

For these models, begin by creating features.

Feature Engineering

Transformers

The package provides many transformers for time series data preprocessing and featurization. The examples that follow demonstrate some of the preprocessing and featurization functionality.

```

# DropColumns: Drop columns that should not be included for modeling. `logmove` is the log of the number of
# units sold, so providing this number would be cheating. `WeekFirstDay` would be
# redundant since we already have a feature for the last day of the week.
columns_to_drop = ['logmove', 'WeekFirstDay', 'week']
column_dropper = DropColumns(columns_to_drop)

# TimeSeriesImputer: Fill missing values in the features
# First, we need to create a dictionary with key as column names and value as values used to fill missing
# values for that column. We are going to use the mean to fill missing values for each column.
columns_with_missing_values =
train_imputed_tsdf.columns[pd.DataFrame(train_imputed_tsdf).isnull().any()].tolist()
columns_with_missing_values = [c for c in columns_with_missing_values if c not in columns_to_drop]
missing_value_imputation_dictionary = {}
for c in columns_with_missing_values:
    missing_value_imputation_dictionary[c] = train_imputed_tsdf[c].mean()
fillna_imputer = TimeSeriesImputer(option='fillna',
                                    input_column=columns_with_missing_values,
                                    value=missing_value_imputation_dictionary)

# TimeIndexFeaturizer: extract temporal features from timestamps
time_index_featurizer = TimeIndexFeaturizer(correlation_cutoff=0.1, overwrite_columns=True)

# GrainIndexFeaturizer: create indicator variables for stores and brands
grain_featurizer = GrainIndexFeaturizer(overwrite_columns=True, ts_frequency=obj_series_freq)

```

Pipelines

Pipeline objects make it easy to save a set of steps so they can be applied over and over again to different objects. Also, pipeline objects can be pickled to make them easily portable to other machines for deployment. You can chain all the transformers you've created so far using a pipeline.

```

pipeline_ml = AzureMLForecastPipeline([('drop_columns', column_dropper),
                                       ('fillna_imputer', fillna_imputer),
                                       ('time_index_featurizer', time_index_featurizer),
                                       ('grain_featurizer', grain_featurizer)
                                       ])

train_feature_tsdf = pipeline_ml.fit_transform(train_imputed_tsdf)
test_feature_tsdf = pipeline_ml.transform(test_tsdf)

# Let's get a look at our new feature set
print(train_feature_tsdf.head())

```

```

F1 2018-06-14 23:10:03,472 INFO azureml.timeseries - pipeline fit_transform started.
F1 2018-06-14 23:10:07,317 INFO azureml.timeseries - pipeline fit_transform finished. Time elapsed
0:00:03.845078
F1 2018-06-14 23:10:07,317 INFO azureml.timeseries - pipeline transforms started.
F1 2018-06-14 23:10:16,499 INFO azureml.timeseries - pipeline transforms finished. Time elapsed 0:00:09.182314
                                         feat  price  AGE60  EDUC  ETHNIC  \
WeekLastDay      store brand
1990-06-20 23:59:59 2    dominicks  1.00  1.59  0.23  0.25  0.11
                           minute.maid  0.00  3.17  0.23  0.25  0.11
                           tropicana  0.00  3.87  0.23  0.25  0.11
                           5    dominicks  1.00  1.59  0.12  0.32  0.05
                           minute.maid  0.00  2.99  0.12  0.32  0.05

                                         INCOME  HHLARGE  WORKWOM  HVAL150  \
WeekLastDay      store brand
1990-06-20 23:59:59 2    dominicks  10.55  0.10  0.30  0.46
                           minute.maid  10.55  0.10  0.30  0.46
                           tropicana  10.55  0.10  0.30  0.46
                           5    dominicks  10.92  0.10  0.41  0.54
                           minute.maid  10.92  0.10  0.41  0.54

                                         SSTRDIST  ...  CPWVOL5  \
WeekLastDay      store brand
1990-06-20 23:59:59 2    dominicks  2.11  ...  0.38
                           minute.maid  2.11  ...  0.38
                           tropicana  2.11  ...  0.38
                           5    dominicks  3.80  ...  0.74
                           minute.maid  3.80  ...  0.74

                                         Quantity  TEMP  DEWP  WDSP  PRCP  year  \
WeekLastDay      store brand
1990-06-20 23:59:59 2    dominicks  10560.00 72.00 61.87 9.74 0.19 1990
                           minute.maid  4480.00 72.00 61.87 9.74 0.19 1990
                           tropicana  8256.00 72.00 61.87 9.74 0.19 1990
                           5    dominicks  1792.00 72.00 61.87 9.74 0.19 1990
                           minute.maid  4224.00 72.00 61.87 9.74 0.19 1990

                                         day  grain_brand  grain_store
WeekLastDay      store brand
1990-06-20 23:59:59 2    dominicks  20    dominicks      2
                           minute.maid  20    minute.maid      2
                           tropicana  20    tropicana      2
                           5    dominicks  20    dominicks      5
                           minute.maid  20    minute.maid      5

```

[5 rows x 22 columns]

RegressionForecaster

The [RegressionForecaster](#) function wraps sklearn regression estimators so that they can be trained on `TimeSeriesDataFrame`. The wrapped forecaster also puts each group, in this case store, into the same model. The forecaster can learn one model for a group of series that were deemed similar and can be pooled together. One model for a group of series often uses the data from longer series to improve forecasts for short series. You can substitute these models for any other models in the library that support regression.

```

lasso_model = RegressionForecaster(estimator=Lasso(),
                                     make_grain_features=False)
elastic_net_model = RegressionForecaster(estimator=ElasticNet(),
                                         make_grain_features=False)
knn_model = RegressionForecaster(estimator=KNeighborsRegressor(),
                                 make_grain_features=False)
random_forest_model = RegressionForecaster(estimator=RandomForestRegressor(),
                                         make_grain_features=False)
boosted_trees_model = RegressionForecaster(estimator=GradientBoostingRegressor(),
                                         make_grain_features=False)

ml_union = ForecasterUnion(forecaster_list=[
    ('lasso', lasso_model),
    ('elastic_net', elastic_net_model),
    ('knn', knn_model),
    ('random_forest', random_forest_model),
    ('boosted_trees', boosted_trees_model)
])

```

```

ml_union.fit(train_feature_tsdf, y=train_feature_tsdf.ts_value)
ml_results = ml_union.predict(test_feature_tsdf, retain_feature_column=True)

```

```

ml_model_MAPE = ml_results.calc_error(err_name='MAPE', by='ModelName')
ml_model_MedianAPE = ml_results.calc_error(err_name='MedianAPE',
                                           err_fun=calc_median_ape,
                                           by='ModelName')
ml_model_errors = ml_model_MAPE.merge(ml_model_MedianAPE, on='ModelName')
all_errors = pd.concat([univariate_model_errors, ml_model_errors])
all_errors.sort_values('MedianAPE')

```

	MODELNAME	MAPE	MEDIANAPE
4	random_forest	78.82	42.81
0	boosted_trees	78.46	45.37
2	naive	103.57	59.14
2	knn	129.85	65.37
1	elastic_net	125.11	65.59
3	seasonal_naive	180.54	65.99
0	arima	126.57	66.49
3	lasso	112.87	67.92
1	ets	187.89	75.73

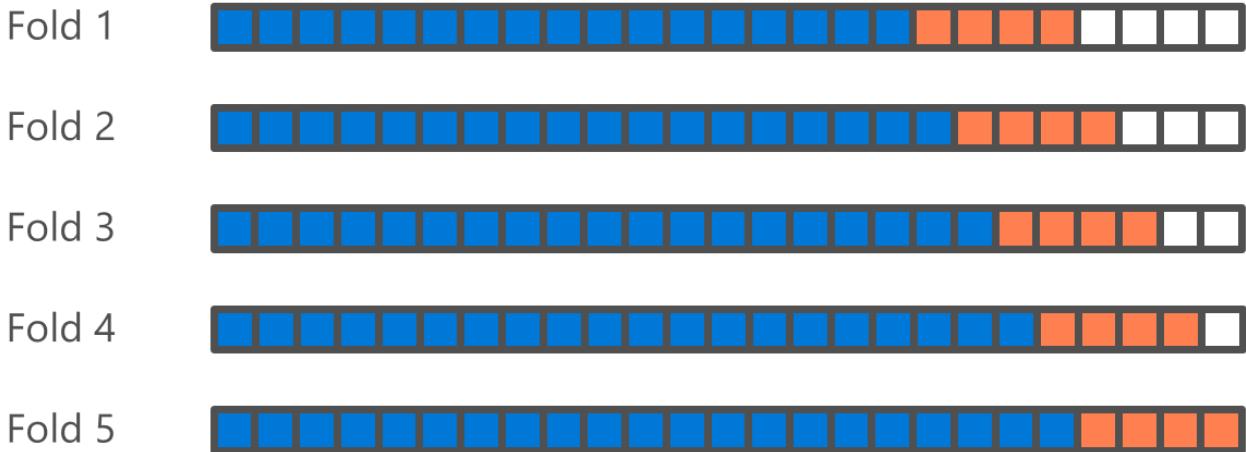
Some machine learning models were able to take advantage of the added features and the similarities between series to get better forecast accuracy.

Cross Validation, Parameter, and Model Sweeping

The package adapts some traditional machine learning functions for a forecasting application.

[RollingOriginValidator](#) does cross-validation temporally, respecting what would and would not be known in a forecasting framework.

In the figure below, each square represents data from one time point. The blue squares represent training and orange squares represent testing in each fold. Testing data must come from the time points after the largest training time point. Otherwise, future data is leaked into training data causing the model evaluation to become invalid.



Parameter Sweeping

The [TSGridSearchCV](#) class exhaustively searches over specified parameter values and uses [RollingOriginValidator](#) to evaluate parameter performance in order to find the best parameters.

```
# Set up the `RollingOriginValidator` to do 2 folds of rolling origin cross-validation
rolcv = RollingOriginValidator(n_splits=2)
randomforest_model_for_cv = RegressionForecaster(estimator=RandomForestRegressor(),
                                                 make_grain_features=False)

# Set up our parameter grid and feed it to our grid search algorithm
param_grid_rf = {'estimator__n_estimators': np.array([10, 50, 100])}
grid_cv_rf = TSGridSearchCV(randomforest_model_for_cv, param_grid_rf, cv=rolcv)

# fit and predict
randomforest_cv_fitted= grid_cv_rf.fit(train_feature_tsdf, y=train_feature_tsdf.ts_value)
print('Best parameter: {}'.format(randomforest_cv_fitted.best_params_))
```

```
Best parameter: {'estimator__n_estimators': 100}
```

Model Sweeping

The [BestOfForecaster](#) class selects the model with the best performance from a list of given models. Similar to [TSGridSearchCV](#), it also uses RollingOriginValidator for cross validation and performance evaluation.

Here we pass a list of two models to demonstrate the usage of [BestOfForecaster](#)

```
best_of_forecaster = BestOfForecaster(forecaster_list=[('naive', naive_model),
                                                       ('random_forest', random_forest_model)])
best_of_forecaster_fitted = best_of_forecaster.fit(train_feature_tsdf,
                                                   validator=RollingOriginValidator(n_step=20,
                                                       max_horizon=40))
best_of_forecaster_prediction = best_of_forecaster_fitted.predict(test_feature_tsdf)
best_of_forecaster_prediction.head()
```

					POINTFORECAST	DISTRIBUTIONFORECAST	QUANTITY
WEEKLASTDAY	STORE	BRAND	FORECASTORIGINTIME	MODELNAME			
1992-01-08 23:59:59	2	DOMINICKS	1992-01-01 23:59:59	RANDOM_FOREST	9299.20	<scipy.stats._distn_infrastructure.rv_frozen o...	11712.00
1992-01-15 23:59:59	2	DOMINICKS	1992-01-01 23:59:59	RANDOM_FOREST	10259.20	<scipy.stats._distn_infrastructure.rv_frozen o...	4032.00
1992-01-22 23:59:59	2	DOMINICKS	1992-01-01 23:59:59	RANDOM_FOREST	6828.80	<scipy.stats._distn_infrastructure.rv_frozen o...	6336.00
1992-01-29 23:59:59	2	DOMINICKS	1992-01-01 23:59:59	RANDOM_FOREST	16633.60	<scipy.stats._distn_infrastructure.rv_frozen o...	13632.00
1992-02-05 23:59:59	2	DOMINICKS	1992-01-01 23:59:59	RANDOM_FOREST	12774.40	<scipy.stats._distn_infrastructure.rv_frozen o...	45120.00

Build the final pipeline

Now that you have identified the best model, you can build and fit your final pipeline with all transformers and the best model.

```
random_forest_model_final =
RegressionForecaster(estimator=RandomForestRegressor(100),make_grain_features=False)
pipeline_ml.add_pipeline_step('random_forest_estimator', random_forest_model_final)
pipeline_ml_fitted = pipeline_ml.fit(train_imputed_tsdf)
final_prediction = pipeline_ml_fitted.predict(test_tsdf)
final_median_ape = final_prediction.calc_error(err_name='MedianAPE', err_fun=calc_median_ape)
print('Median of APE of final pipeline: {0}'.format(final_median_ape))
```

```
F1 2018-05-04 11:07:04,108 INFO azureml.timeseries - pipeline fit started.
F1 2018-05-04 11:07:43,121 INFO azureml.timeseries - pipeline fit finished. Time elapsed 0:00:39.012880
F1 2018-05-04 11:07:43,136 INFO azureml.timeseries - pipeline predict started.
F1 2018-05-04 11:08:03,564 INFO azureml.timeseries - pipeline predict finished. Time elapsed 0:00:20.428147
Median of APE of final pipeline: 42.54336821266968
```

Visualization

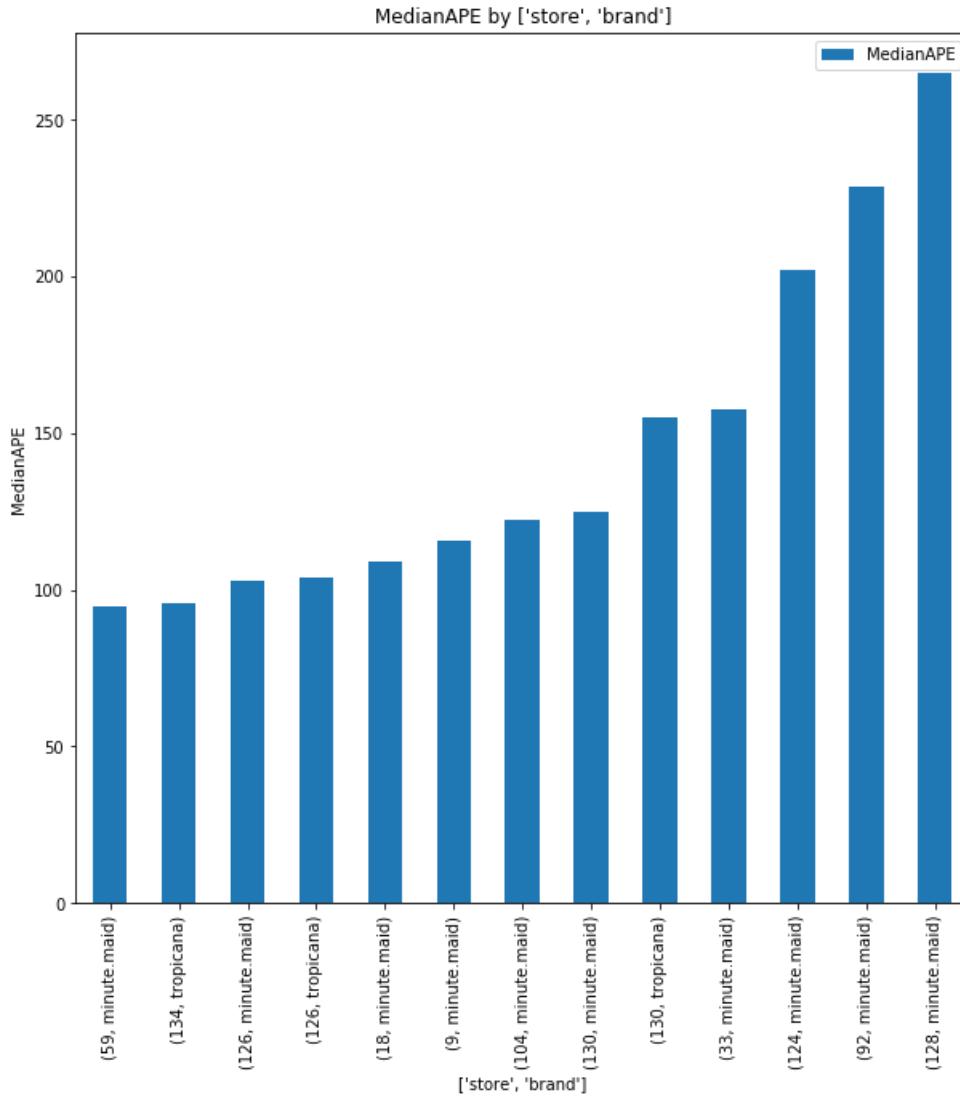
The `ForecastDataFrame` class provides plotting functions for visualizing and analyzing forecasting results. Use the commonly used charts with your data. Please see the sample notebook below on plotting functions for all the functions available.

The `show_error` function plots performance metrics aggregated by an arbitrary column. By default, the

`show_error` function aggregates by the `grain_colnames` of the `ForecastDataFrame`. It's often useful to identify the grains/groups with the best or worst performance, especially when you have a large number of time series. The `performance_percent` argument of `show_error` allows you to specify a performance interval and plot the error of a subset of grains/groups.

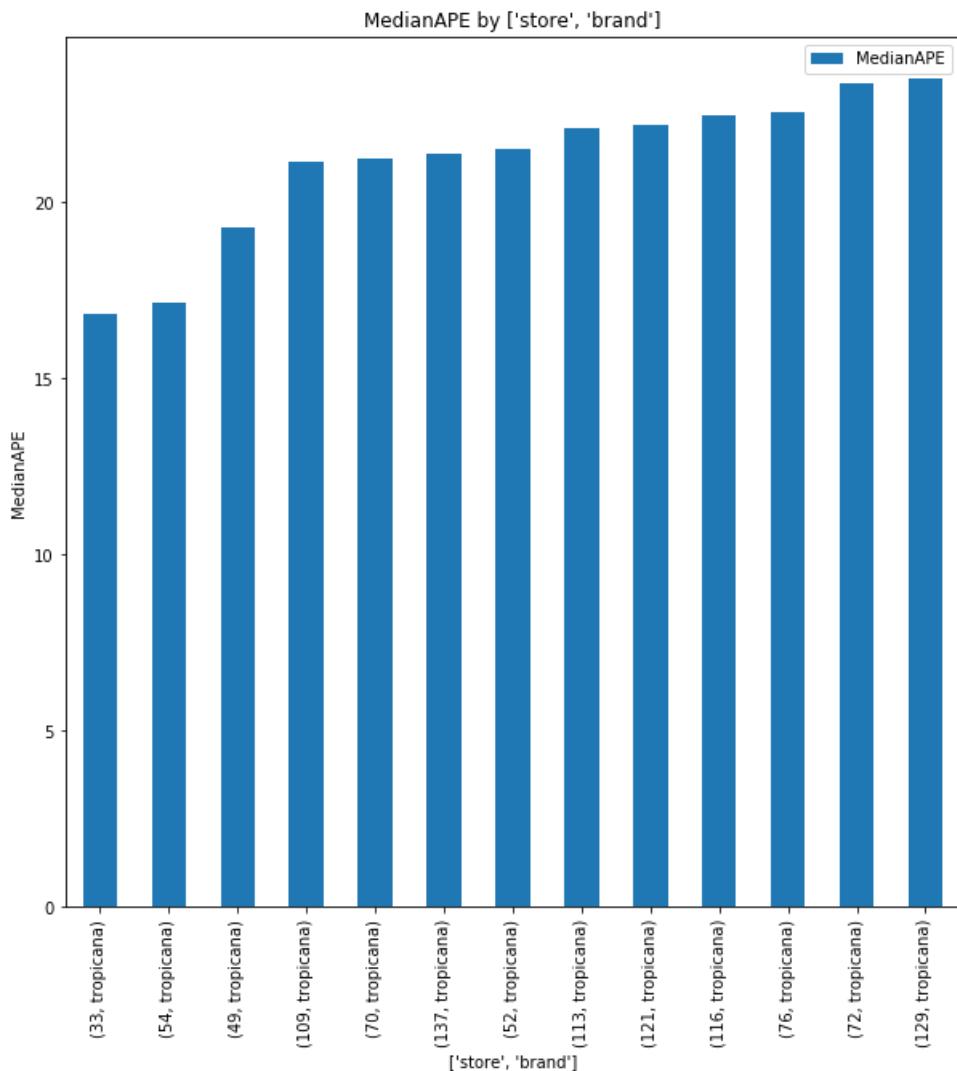
Plot the grains with the bottom 5% performance, i.e. top 5% MedianAPE

```
fig, ax = best_of_forecaster_prediction.show_error(err_name='MedianAPE', err_fun=calc_median_ape,
performance_percent=(0.95, 1))
```



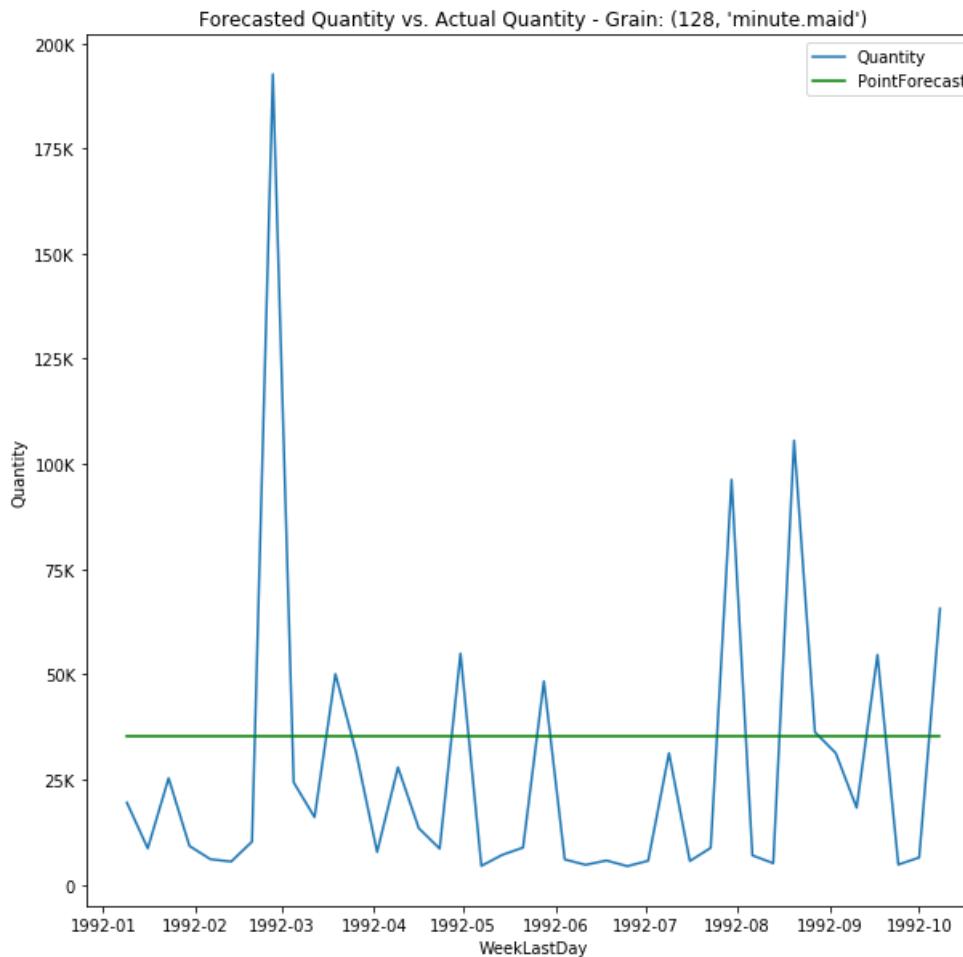
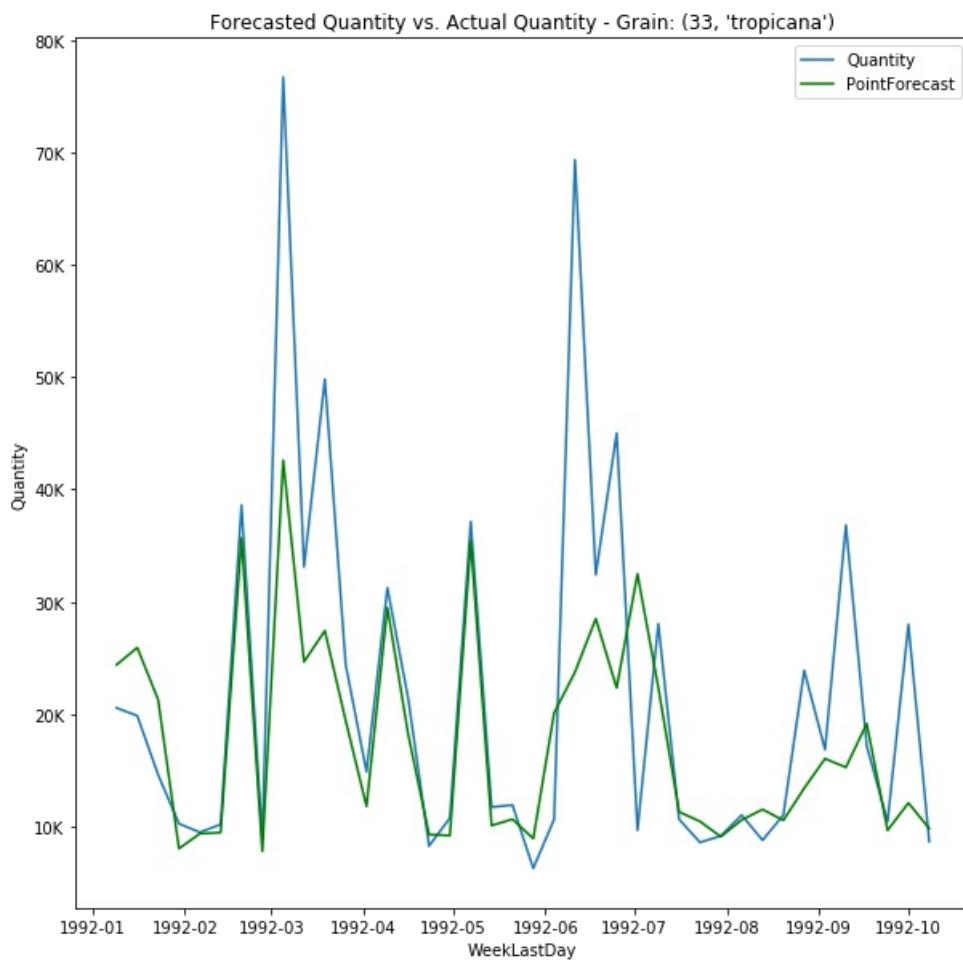
Plot the grains with the top 5% of performance, i.e. bottom 5% MedianAPE.

```
fig, ax = best_of_forecaster_prediction.show_error(err_name='MedianAPE', err_fun=calc_median_ape,
performance_percent=(0, 0.05))
```



Once you have an idea of the overall performance, you may want to explore individual grains, especially those that performed poorly. The `plot_forecast_by_grain` method plots forecast vs. actual of specified grains. Here, we plot the grain with the best performance and the grain with the worst performance discovered in the `show_error` plot.

```
fig_ax = best_of_forecaster_prediction.plot_forecast_by_grain(grains=[(33, 'tropicana'), (128, 'minute.maid')])
```



Additional Notebooks

For a deeper dive on the major features of AMLPF, please refer to the following notebooks with more details and examples of each feature:

[Notebook on TimeSeriesDataFrame](#)

[Notebook on Data Wrangling](#)

[Notebook on Transformers](#)

[Notebook on Models](#)

[Notebook on Cross Validation](#)

[Notebook on Lag Transformer and OriginTime](#)

[Notebook on Plotting Functions](#)

Operationalization

In this section, you deploy a pipeline as an Azure Machine Learning web service and consume it for training and scoring. Currently, only pipelines there are not fitted are supported for deployment. Scoring the deployed web service retrains the model and generates forecasts on new data.

Set model deployment parameters

Change the following parameters to your own values. Make sure your Azure Machine Learning environment, model management account, and resource group are located in the same region.

```
azure_subscription = '<subscription name>'  
  
# Two deployment modes are supported: 'local' and 'cluster'.  
# 'local' deployment deploys to a local docker container.  
# 'cluster' deployment deploys to a Azure Container Service Kubernetes-based cluster  
deployment_type = '<deployment mode>'  
  
# The deployment environment name.  
# This could be an existing environment or a new environment to be created automatically.  
aml_env_name = '<deployment env name>'  
  
# The resource group that contains the Azure resources related to the AML environment.  
aml_env_resource_group = '<env resource group name>'  
  
# The location where the Azure resources related to the AML environment are located at.  
aml_env_location = '<env resource location>'  
  
# The AML model management account name. This could be an existing model management account a new model  
management  
# account to be created automatically.  
model_management_account_name = '<model management account name>'  
  
# The resource group that contains the Azure resources related to the model management account.  
model_management_account_resource_group = '<model management account resource group>'  
  
# The location where the Azure resources related to the model management account are located at.  
model_management_account_location = '<model management account location>'  
  
# The name of the deployment/web service.  
deployment_name = '<web service name>'  
  
# The directory to store deployment related files, such as pipeline pickle file, score script,  
# and conda dependencies file.  
deployment_working_directory = '<local working directory>'
```

Define the Azure Machine Learning environment and deployment

```

aml_settings = AMLSettings(azure_subscription=azure_subscription,
                           env_name=aml_env_name,
                           env_resource_group=aml_env_resource_group,
                           env_location=aml_env_location,
                           model_management_account_name=model_management_account_name,
                           model_management_account_resource_group=model_management_account_resource_group,
                           model_management_account_location=model_management_account_location,
                           cluster=deployment_type)

random_forest_model_deploy = RegressionForecaster(estimator=RandomForestRegressor(), make_grain_features=False)
pipeline_deploy = AzureMLForecastPipeline([('drop_columns', column_dropper),
                                           ('fillna_imputer', fillna_imputer),
                                           ('time_index_featurizer', time_index_featurizer),
                                           ('random_forest_estimator', random_forest_model_deploy)
                                         ])

aml_deployment = ForecastWebserviceFactory(deployment_name=deployment_name,
                                             aml_settings=aml_settings,
                                             pipeline=pipeline_deploy,
                                             deployment_working_directory=deployment_working_directory,
                                             ftk_wheel_loc='https://azurermftkrelease.blob.core.windows.net/dailyrelease/azurermftk-0.1.18165.29a1-py3-none-any.whl')

```

Create the web service

```

# This step can take 5 to 20 minutes
aml_deployment.deploy()

```

Score the web service

To score a small dataset, use the [score](#) method to submit one web service call for all the data.

```

# Need to add empty prediction columns to the validation data frame and create a ForecastDataFrame.
# The scoring API will be updated in later versions to take TimeSeriesDataFrame directly.
validate_tsdf = test_tsdf.assign(PointForecast=0.0, DistributionForecast=np.nan)
validate_fcast = ForecastDataFrame(validate_tsdf, pred_point='PointForecast',
                                   pred_dist='DistributionForecast')

# Define Score Context
score_context = ScoreContext(input_training_data_tsdf=train_imputed_tsdf,
                             input_scoring_data_fcdf=validate_fcast,
                             pipeline_execution_type='train_predict')

# Get deployed web service
aml_web_service = aml_deployment.get_deployment()

# Score the web service
results = aml_web_service.score(score_context=score_context)

```

To score a large dataset, use the [parallel scoring](#) mode to submit multiple web service calls, one for each group of data.

```

results = aml_web_service.score(score_context=score_context, method='parallel')

```

Next steps

Learn more about the Azure Machine Learning Package for Forecasting in these articles:

- Read the [package overview](#).

- Explore the [reference docs](#) for this package.
- Learn about [other Python packages for Azure Machine Learning](#).

Azure Machine Learning Hardware Acceleration package

12/11/2018 • 2 minutes to read • [Edit Online](#)

NOTE

This article is deprecated. This FPGA package was deprecated. Support for this functionality was added to the Azure ML SDK. Support for this package will end incrementally. [View the support timeline](#). Learn about updated [FPGA support](#).

The Azure Machine Learning Hardware Acceleration package is a Python pip-installable extension for Azure Machine Learning that enables data scientists and AI developers to quickly:

- Featurize images with a quantized version of ResNet 50
- Train classifiers based on those features
- Deploy models to [field programmable gate arrays \(FPGA\)](#) on Azure for ultra-low latency inferencing

Prerequisites

NOTE

If you don't have an Azure subscription, create a free account before you begin. Try the [free or paid version of Azure Machine Learning service](#) today.

1. An Azure Machine Learning Model Management account. For more information on creating the account, see the [Azure Machine Learning Quickstart and Workbench installation](#) document.
2. The package must be installed.

How to install the package

1. Download and install the latest version of [Git](#).
2. Install [Anaconda \(Python 3.6\)](#)

To download a pre-configured Anaconda environment, use the following command from the Git prompt:

```
git clone https://aka.ms/aml-real-time-ai
```

3. To create the environment, open an **Anaconda Prompt** and use the following command:

```
conda env create -f aml-real-time-ai/environment.yml
```

4. To activate the environment, use the following command:

```
conda activate amlrealtimeai
```

Sample code

This sample code walks you through using the SDK to deploy a model to an FPGA.

1. Import the package:

```
import amlrealtimeai
from amlrealtimeai import resnet50
```

2. Pre-process the image:

```
from amlrealtimeai.resnet50.model import LocalQuantizedResNet50
model_path = os.path.expanduser('~/models')
model = LocalQuantizedResNet50(model_path)
print(model.version)
```

3. Featurize the images:

```
from amlrealtimeai.resnet50.model import LocalQuantizedResNet50
model_path = os.path.expanduser('~/models')
model = LocalQuantizedResNet50(model_path)
print(model.version)
```

4. Create a classifier:

```
model.import_graph_def(include_featurizer=False)
print(model.classifier_input)
print(model.classifier_output)
```

5. Create the service definition:

```
from amlrealtimeai.pipeline import ServiceDefinition, TensorflowStage, BrainWaveStage
save_path = os.path.expanduser('~/models/save')
service_def_path = os.path.join(save_path, 'service_def.zip')

service_def = ServiceDefinition()
service_def.pipeline.append(TensorflowStage(tf.Session(), in_images, image_tensors))
service_def.pipeline.append(BrainWaveStage(model))
service_def.pipeline.append(TensorflowStage(tf.Session(), model.classifier_input,
model.classifier_output))
service_def.save(service_def_path)
print(service_def_path)
```

6. Prepare the model to run on an FPGA:

```
from amlrealtimeai import DeploymentClient

subscription_id = "<Your Azure Subscription ID>"
resource_group = "<Your Azure Resource Group Name>"
model_management_account = "<Your AzureML Model Management Account Name>

model_name = "resnet50-model"
service_name = "quickstart-service"

deployment_client = DeploymentClient(subscription_id, resource_group, model_management_account)
```

7. Deploy the model to run on an FPGA:

```
service = deployment_client.get_service_by_name(service_name)
model_id = deployment_client.register_model(model_name, service_def_path)

if(service is None):
    service = deployment_client.create_service(service_name, model_id)
else:
    service = deployment_client.update_service(service.id, model_id)
```

8. Create the client:

```
from amlrealtimeai import PredictionClient
client = PredictionClient(service.ipAddress, service.port)
```

9. Call the API:

```
image_file = R'C:\path_to_file\image.jpg'
results = client.score_image(image_file)
```

Reporting issues

Use the [forum](#) to report any issues you encounter with the package.

Next steps

[Deploy a model as a web service on an FPGA](#)

Build and deploy text classification models with Azure Machine Learning

11/16/2018 • 15 minutes to read • [Edit Online](#)

NOTE

This article is deprecated. Support for earlier versions of this service will end incrementally. [View the support timeline](#). Start using the latest version with this [quickstart](#).

In this article, you can learn how to train and deploy a text classification model with **Azure Machine Learning Package for Text Analytics** (AMLPTA). The goal of text classification is to assign a piece of text to one or more predefined classes or categories. This text could, for example, be a document, news article, search query, email, tweet, support tickets.

There are broad applications of text classification such as:

- Categorizing newspaper articles and news wire contents into topics
- Organizing web pages into hierarchical categories
- Filtering spam email
- Sentiment analysis
- Predicting user intent from search queries
- Routing support tickets
- Analyzing customer feedback

The text classification model building and deployment workflow for a model with AMLPTA is as follows:

1. Load the data
2. Train the model
3. Apply the classifier
4. Evaluate model performance
5. Save the pipeline
6. Test the pipeline
7. Deploy the model as a web service

Consult the [package reference documentation](#) for the detailed reference for each module and class.

The sample code in this article uses a scikit-learn pipeline.

Prerequisites

1. If you don't have an Azure subscription, create a [free account](#) before you begin.
2. The following accounts and application must be set up and installed:
 - Azure Machine Learning Experimentation account
 - An Azure Machine Learning Model Management account
 - Azure Machine Learning Workbench installed

If these three are not yet created or installed, follow the [Azure Machine Learning Quickstart and Workbench installation](#) article.

3. The Azure Machine Learning Package for Text Analytics must be installed. Learn how to [install this package here](#).

Sample data and Jupyter notebook

Get the Jupyter notebook

Try it out yourself. Download the notebook and run it yourself.

[Get the Jupyter notebook](#)

Download and Explore the sample data

The following example uses the [20 newsgroups dataset](#) that is available through the scikit-learn library to demonstrate how to create a text classifier with Azure Machine Learning Package for Text Analytics.

The 20 newsgroups dataset has around 18,000 newsgroups posts on 20 different topics divided into two subsets: one for training and the other one for performance evaluation. The split into train and test is based upon each message post date whether before or after a specific date.

```
# Import Packages
# Use Azure Machine Learning history magic to control history collection
# History is off by default, options are "on", "off", or "show"
#%azureml history on
%matplotlib inline
# Use the Azure Machine Learning data collector to log various metrics
from azureml.logging import get_azureml_logger
import os

logger = get_azureml_logger()

# Log cell runs into run history
logger.log('Cell','Set up run')
# from tatk.utils import load_newsgroups_data, data_dir, dictionaries_dir, models_dir
import pip
pip.main(["show", "azureml-tatk"])
```

Set the location of the data

Set the location where you have downloaded the data in the data dir parameter. You can also use your own data, the input dataset must be a *.tsv file format.

```

import os
import pandas as pd

#set the working directory where to save the training data files
resources_dir = os.path.join(os.path.expanduser("~/"), "tatk", "resources")
data_dir = os.path.join(os.path.expanduser("~/"), "tatk", "data")

from sklearn.datasets import fetch_20newsgroups
twenty_train = fetch_20newsgroups(data_home=data_dir, subset='train')
X_train, y_train = twenty_train.data, twenty_train.target
df_train = pd.DataFrame({"text":X_train, "label":y_train})

twenty_test = fetch_20newsgroups(data_home=data_dir, subset='test')
X_test, y_test = twenty_test.data, twenty_test.target
df_test = pd.DataFrame({"text":X_test, "label":y_test})

# Training Dataset Location
#training_file_path = <specify-your-own-training-data-file-path-here>
# df_train = pd.read_csv(training_file_path,
# #                         sep = '\t',
# #                         header = 0, names= <specify-your-column-name-list-here>)
df_train.head()
print("df_train.shape= {}".format(df_train.shape))

# Test Dataset Location
#test_file_path = <specify-your-own-test-data-file-path-here>
# df_test = pd.read_csv(test_file_path,
# #                         sep = '\t',
# #                         header = 0, names= <specify-your-column-name-list-here>)

print("df_test.shape= {}".format(df_test.shape))
df_test.head()

```

```

df_train.shape= (11314, 2)
df_test.shape= (7532, 2)

```

The data consists of label and text

	LABEL	TEXT
0	7	From: v064mb9k@ubvmsd.cc.buffalo.edu (NEIL B. ...)
1	5	From: Rick Miller <rick@ee.uwm.edu>\nSubject: ...
2	0	From: mathew <mathew@mantis.co.uk>\nSubject: R...
3	17	From: bakken@cs.arizona.edu (Dave Bakken)\nSub...
4	19	From: livesey@solntze.wpd.sgi.com (Jon Livesey...)

Get the correspondance between categories and their name.

```
int_to_categories = pd.DataFrame({'category':range(20), 'category_name': list(twenty_train.target_names)})  
int_to_categories
```

	CATEGORY	CATEGORY_NAME
0	0	alt.atheism
1	1	comp.graphics
2	2	comp.os.ms-windows.misc
3	3	comp.sys.ibm.pc.hardware
4	4	comp.sys.mac.hardware
5	5	comp.windows.x
6	6	misc.forsale
7	7	rec.autos
8	8	rec.motorcycles
9	9	rec.sport.baseball
10	10	rec.sport.hockey
11	11	sci.crypt
12	12	sci.electronics
13	13	sci.med
14	14	sci.space
15	15	soc.religion.christian
16	16	talk.politics.guns
17	17	talk.politics.mideast
18	18	talk.politics.misc
19	19	talk.religion.misc

Now, you can create a preliminary exploration plot histogram of the class frequency in training and test data sets.

```

import numpy as np
import math
from matplotlib import pyplot as plt

data = df_train["label"].values
labels = set(data)
print(labels)
bins = range(len(labels)+1)

#plt.xlim([min(data)-5, max(data)+5])

plt.hist(data, bins=bins, alpha=0.8)
plt.title('training data distribution over the class labels')
plt.xlabel('class label')
plt.ylabel('frequency')
plt.grid(True)
plt.show()

data = df_test["label"].values
labels = set(data)
print(labels)
bins = range(len(labels)+1)

#plt.xlim([min(data)-5, max(data)+5])

plt.hist(data, bins=bins, alpha=0.8)
plt.title('test data distribution over the class labels')
plt.xlabel('class label')
plt.ylabel('frequency')
plt.grid(True)
plt.show()

```

When running the Jupyter notebook, plots are displayed after the preceding code block is run.

Train the model

Specify scikit-learn algorithm and define the text classifier

This step involves training a scikit-learn text classification model using One-versus-Rest Logistic Regression learning algorithm.

For the full list of learnings, refer to the [Scikit Learners](#) documentation.

```

from sklearn.linear_model import LogisticRegression
import tatk
from tatk.pipelines.text_classification.text_classifier import TextClassifier

log_reg_learner = LogisticRegression(penalty='l2', dual=False, tol=0.0001,
                                      C=1.0, fit_intercept=True, intercept_scaling=1,
                                      class_weight=None, random_state=None,
                                      solver='lbfgs', max_iter=100, multi_class='ovr',
                                      verbose=1, warm_start=False, n_jobs=3)

#train the model a text column "tweets"
text_classifier = TextClassifier(estimator=log_reg_learner,
                                 text_cols = ["text"],
                                 label_cols = ["label"],
                                 # numeric_cols = None,
                                 # cat_cols = None,
                                 extract_word_ngrams=True, extract_char_ngrams=True)

```

```

TextClassifier::create_pipeline ==> start
:: number of jobs for the pipeline : 12
0  text_nltk_preprocessor
1  text_word_ngrams
2  text_char_ngrams
3  assembler
4  learner
TextClassifier::create_pipeline ==> end

```

Fit the model

Use the default parameters of the package. By default, the text classifier extracts:

- Word unigrams and bigrams
- Character 4 grams

```
text_classifier.fit(df_train)
```

```

TextClassifier::fit ==> start
schema: col=label:I4:0 col=text:TX:1 header+
NltkPreprocessor::tatk_fit_transform ==> start
NltkPreprocessor::tatk_fit_transform ==> end      Time taken: 0.08 mins
NGramsVectorizer::tatk_fit_transform ==> startNGramsVectorizer::tatk_fit_transform ==> start

    vocabulary size=216393
NGramsVectorizer::tatk_fit_transform ==> end      Time taken: 0.41 mins
    vocabulary size=67230
NGramsVectorizer::tatk_fit_transform ==> end      Time taken: 0.49 mins
VectorAssembler::transform ==> start, num of input records=11314
(11314, 216393)
(11314, 67230)
all_features:
(11314, 283623)
Time taken: 0.06 mins
VectorAssembler::transform ==> end
LogisticRegression::tatk_fit ==> start

[Parallel(n_jobs=3)]: Done 20 out of 20 | elapsed: 2.4min finished

LogisticRegression::tatk_fit ==> end      Time taken: 2.4 mins
Time taken: 3.04 mins
TextClassifier::fit ==> end

TextClassifier(add_index_col=False, callable_proprocessors_list=None,
  cat_cols=None, char_hashing_original=False, col_prefix='tmp_00_',
  decompose_n_grams=False, detect_phrases=False,
  dictionary_categories=None, dictionary_file_path=None,
  embedding_file_path=None, embedding_file_path_fastText=None,
  estimator=None, estimator_vectorizers_list=None,
  extract_char_ngrams=True, extract_word_ngrams=True,
  label_cols=['label'], numeric_cols=None,
  pos_tagger_vectorizer=False,
  preprocessor_dictionary_file_path=None, regex_replacement='',
  replace_regex_pattern=None, scale_numeric_cols=False,
  text_callable_list=None, text_cols=['text'], text_regex_list=None,
  weight_col=None)

```

During training, you must have both text and label columns. While, only the text column is needed for predictions.

Examine and set the parameters of the different pipeline steps

Typically, you set the parameters before you fit a model.

Example shown with `text_word_ngrams`

The following code samples show you how to train the model using the default pipeline and model parameters.

To see what parameters are included for "text_word_ngrams", use [get_step_param_names_by_name](#). This function returns the parameters such as lowercase, input_col, output_col and so on.

```
text_classifier.get_step_param_names_by_name("text_word_ngrams")
```

```
['min_df',
 'strip_accents',
 'max_df',
 'decode_error',
 'max_features',
 'binary',
 'input',
 'vocabulary',
 'analyzer',
 'token_pattern',
 'encoding',
 'use_idf',
 'save_overwrite',
 'output_col',
 'stop_words',
 'sublinear_tf',
 'input_col',
 'lowercase',
 'ngram_range',
 'preprocessor',
 'tokenizer',
 'hashing',
 'dtype',
 'norm',
 'smooth_idf',
 'n_hashing_features']
```

Next, check the parameter values for "text_char_ngrams":

```
text_classifier.get_step_params_by_name("text_char_ngrams")
```

```
{'analyzer': 'char_wb',
'binary': False,
'decode_error': 'strict',
'dtype': numpy.float32,
'encoding': 'utf-8',
'hashing': False,
'input': 'content',
'input_col': 'NltkPreprocessor5283a730506549cc880f074e750607b0',
'lowercase': True,
'max_df': 1.0,
'max_features': None,
'min_df': 3,
'n_hashing_features': None,
'ngram_range': (4, 4),
'norm': 'l2',
'output_col': 'NGramsVectorizer8eb11031f6b64eaaad9ff0fd3b0f5b80',
'preprocessor': None,
'save_overwrite': True,
'smooth_idf': True,
'stop_words': None,
'strip_accents': None,
'sublinear_tf': False,
'token_pattern': '(?u)\\b\\\\w\\\\w+\\\\b',
'tokenizer': None,
'use_idf': True,
'vecocabulary': None}
```

If necessary, you can change the default parameters. With the following code, you can change the range of extracted character n-grams from (4,4) to (3,4) to extract both character tri-grams and 4 grams:

```
text_classifier.set_step_params_by_name("text_char_ngrams", ngram_range =(3,4))
text_classifier.get_step_params_by_name("text_char_ngrams")
```

```
{'analyzer': 'char_wb',
'binary': False,
'decode_error': 'strict',
'dtype': numpy.float32,
'encoding': 'utf-8',
'hashing': False,
'input': 'content',
'input_col': 'NltkPreprocessor5283a730506549cc880f074e750607b0',
'lowercase': True,
'max_df': 1.0,
'max_features': None,
'min_df': 3,
'n_hashing_features': None,
'ngram_range': (3, 4),
'norm': 'l2',
'output_col': 'NGramsVectorizer8eb11031f6b64eaaad9ff0fd3b0f5b80',
'preprocessor': None,
'save_overwrite': True,
'smooth_idf': True,
'stop_words': None,
'strip_accents': None,
'sublinear_tf': False,
'token_pattern': '(?u)\\b\\\\w\\\\w+\\\\b',
'tokenizer': None,
'use_idf': True,
'vecocabulary': None}
```

Export the parameters to a file

If needed, you can optimize model performance by rerunning the model fitting step with revised parameters:

```

import os
params_file_path = os.path.join(data_dir, "params.tsv")
text_classifier.export_params(params_file_path)

```

Apply the classifier

Apply the trained text classifier on the test dataset to generate class predictions:

```
df_test = text_classifier.predict(df_test)
```

```

TextClassifier ::predict ==> start
NltkPreprocessor::tatk_transform ==> start
NltkPreprocessor::tatk_transform ==> end      Time taken: 0.05 mins
NGramsVectorizer::tatk_transform ==> startNGramsVectorizer::tatk_transform ==> start

NGramsVectorizer::tatk_transform ==> end      Time taken: 0.15 mins
NGramsVectorizer::tatk_transform ==> end      Time taken: 0.37 mins
VectorAssembler::transform ==> start, num of input records=7532
(7532, 216393)
(7532, 67230)
all_features::
(7532, 283623)
Time taken: 0.03 mins
VectorAssembler::transform ==> end
LogisticRegression::tatk_predict_proba ==> start
LogisticRegression::tatk_predict_proba ==> end    Time taken: 0.01 mins
LogisticRegression::tatk_predict ==> start
LogisticRegression::tatk_predict ==> end      Time taken: 0.01 mins
Time taken: 0.46 mins
TextClassifier ::predict ==> end
Order of Labels in predicted probabilities saved to attribute label_order of the class object

```

	LABEL	TEXT	PROBABILITIES	PREDICTION
0	7	From: v064mb9k@ubvmsd. cc.buffalo.edu (NEIL B. ...	[0.0165036341329, 0.0548664746458, 0.020549685...]	12
1	5	From: Rick Miller <rick@ee.uwm.edu >\nSubject: ...	[0.025145498995, 0.125877400021, 0.03947047877...]	1
2	0	From: matthew <matthew@mantis.c o.uk>\nSubject: R...	[0.67566338235, 0.0150749738583, 0.00992439163...]	0
3	17	From: bakken@cs.arizona.ed u (Dave Bakken)\nSub...	[0.146063943868, 0.00232465192179, 0.002442807...]	18
4	19	From: livesey@solntze.wpd.s gi.com (Jon Livesey...	[0.670712265297, 0.017332269703, 0.01062429663...]	0

Evaluate model performance

The [evaluation module](#) evaluates the accuracy of the trained text classifier on the test dataset. The evaluate function generates a confusion matrix and provides a macro-F1 score.

```
text_classifier.evaluate(df_test)
```

```
TextClassifier ::evaluate ==> start
Time taken: 0.0 mins
TextClassifier ::evaluate ==> end
```

Plot the confusion without normalization matrix for visualization.

```
evaluator.plot_confusion_matrix(normalize=False,
                                 title='Confusion matrix, without normalization',
                                 print_confusion_matrix=False,
                                 figsize=(8,8),
                                 colors=None)
```

Confusion matrix, without normalization

When running the notebook the confusion matrix will be displayed

Plot the normalized confusion matrix for visualization.

```
evaluator.plot_confusion_matrix(normalize=True,
                                 title='Normalized Confusion matrix',
                                 print_confusion_matrix=False,
                                 figsize=(8,8),
                                 colors=None)
```

```
Normalized confusion matrix
```

When running the notebook the confusion matrix will be displayed

Save the pipeline

Save the classification pipeline into a zip file. Also, save the word-ngrams and character n-grams as text files.

```
import os
working_dir = os.path.join(data_dir, 'outputs')
if not os.path.exists(working_dir):
    os.makedirs(working_dir)

# you can save the trained model as a folder or a zip file
model_file = os.path.join(working_dir, 'sk_model.zip')
text_classifier.save(model_file)
# %azureml upload outputs/models/sk_model.zip
```

```
BaseTextModel::save ==> start
TatkPipeline::save ==> start
Time taken: 0.28 mins
TatkPipeline::save ==> end
Time taken: 0.38 mins
BaseTextModel::save ==> end
```

```
# for debugging, you can save the word n-grams vocabulary to a text file
word_vocab_file_path = os.path.join(working_dir, 'word_ngrams_vocabulary.tsv')
text_classifier.get_step_by_name("text_word_ngrams").save_vocabulary(word_vocab_file_path)
# %azureml upload outputs/dictionaries/word_ngrams_vocabulary.pkl

# for debugging, you can save the character n-grams vocabulary to a text file
char_vocab_file_path = os.path.join(working_dir, 'char_ngrams_vocabulary.tsv')
text_classifier.get_step_by_name("text_char_ngrams").save_vocabulary(char_vocab_file_path)
# %azureml upload outputs/dictionaries/char_ngrams_vocabulary.pkl
```

```
save_vocabulary ==> start
saving 216393 n-grams ...
Time taken: 0.01 mins
save_vocabulary ==> end
save_vocabulary ==> start
saving 67230 n-grams ...
Time taken: 0.0 mins
save_vocabulary ==> end
```

Load the pipeline

Load the classification pipeline and the word-ngrams and character n-grams for inferencing:

```
# in order to deploy the trained model, you have to load the zip file of the classifier pipeline
loaded_text_classifier = TextClassifier.load(model_file)

from tatk.feature_extraction import NGramsVectorizer
word_ngram_vocab = NGramsVectorizer.load_vocabulary(word_vocab_file_path)
char_ngram_vocab = NGramsVectorizer.load_vocabulary(char_vocab_file_path)
```

```
BaseTextModel::load ==> start
TatkPipeline::load ==> start
Time taken: 0.14 mins
TatkPipeline::load ==> end
Time taken: 0.15 mins
BaseTextModel::load ==> end
loading 216393 n-grams ...
loading 67230 n-grams ...
```

Test the pipeline

To evaluate a test dataset, apply the loaded text classification pipeline:

```
predictions = loaded_text_classifier.predict(df_test)
loaded_evaluator = loaded_text_classifier.evaluate(predictions)
loaded_evaluator.get_metrics('macro_f1')
```

```

TextClassifier ::predict ==> start
NltkPreprocessor::tatk_transform ==> start
NltkPreprocessor::tatk_transform ==> end      Time taken: 0.05 mins
NGramsVectorizer::tatk_transform ==> startNGramsVectorizer::tatk_transform ==> start

NGramsVectorizer::tatk_transform ==> end      Time taken: 0.14 mins
NGramsVectorizer::tatk_transform ==> end      Time taken: 0.36 mins
VectorAssembler::transform ==> start, num of input records=7532
(7532, 216393)
(7532, 67230)
all_features::
(7532, 283623)
Time taken: 0.03 mins
VectorAssembler::transform ==> end
LogisticRegression::tatk_predict_proba ==> start
LogisticRegression::tatk_predict_proba ==> end  Time taken: 0.01 mins
LogisticRegression::tatk_predict ==> start
LogisticRegression::tatk_predict ==> end      Time taken: 0.01 mins
Time taken: 0.45 mins
TextClassifier ::predict ==> end
Order of Labels in predicted probabilities saved to attribute label_order of the class object
TextClassifier ::evaluate ==> start
Time taken: 0.0 mins
TextClassifier ::evaluate ==> en

0.82727029243808903

```

Operationalization: deploy and consume

In this section, you deploy the text classification pipeline as an Azure Machine Learning web service using [Azure Machine Learning CLI](#). Then, you consume the web service for training and scoring.

Log in to your Azure subscription with Azure CLI

Using an [Azure](#) account with a valid subscription, log in using the following CLI command:

```
az login
```

- To switch to another Azure subscription, use the command:

```
az account set --subscription [your subscription name]
```

- To see the current model management account, use the command:

```
az ml account modelmanagement show
```

Create and set your deployment environment

You only need to set your deployment environment once. If you don't have one yet, set up your deployment environment now using [these instructions](#).

- Make sure your Azure Machine Learning environment, model management account, and resource group are located in the same region.
- Download the deployment configuration file from Blob storage and save it locally:

```

# Download the deployment config file from Blob storage `url` and save it locally under `file_name`:
deployment_config_file_url =
'https://aztatksa.blob.core.windows.net/dailyrelease/tatk_deploy_config.yaml'
deployment_config_file_path=os.path.join(resources_dir, 'tatk_deploy_config.yaml')
import urllib.request
urllib.request.urlretrieve(deployment_config_file_url, deployment_config_file_path)

```

- Update the deployment configuration file you downloaded to reflect your resources:

```
web_service_name = 'please type your web service name'  
working_directory= os.path.join(resources_dir, 'deployment')  
  
web_service = text_classifier.deploy(web_service_name= web_service_name,  
                                     config_file_path=deployment_config_file_path,  
                                     working_directory= working_directory)
```

4. Given that the trained model is deployed successfully, invoke the scoring web service on new dataset:

```
print("Service URL: {}".format(web_service._service_url))  
print("Service Key: {}".format(web_service._api_key))
```

5. Load the web service at any time using its name:

```
from tatk.operationalization.csi.csi_web_service import CsiWebService  
url = "<please type the service URL here>"  
key = "<please type the service Key here>"  
web_service = CsiWebService(url, key)
```

6. Test the web service with the body of two emails taken from the 20 newsgroups dataset:

```

# Example input data for scoring
import json
dict1 ={}
dict1["recordId"] = "a1"
dict1["data"] = {}
dict1["data"]["text"] = """
I'd be interested in a copy of this code if you run across it.
(Mail to the author bounced)
> / hp1ds1a:comp.graphics / email-address-removed / 12:53 am May 13,
1993 /
> I fooled around with this problem a few years ago, and implemented a
> simple method that ran on a PC.
> was very simple - about 40 or 50 lines of code.
. .
> Somewhere I still have it
> and could dig it out if there was interest.

"""

dict2 ={}
dict2["recordId"] = "b2"
dict2["data"] = {}
dict2["data"]["text"] = """
>> Could the people discussing recreational drugs such as mj, lsd, mdma, etc.,
>> take their discussions to alt.drugs? Their discussions will receive greatest
>> contribution and readership there. The people interested in strictly
>> "smart drugs" (i.e. Nootropics) should post to this group. The two groups
>>(alt.drugs & alt.psychactives) have been used interchangably lately.
>> I do think that alt.psychactives is a deceiving name. alt.psychactives
>> is supposedly the "smart drug" newsgroup according to newsgroup lists on
>> the Usenet. Should we establish an alt.nootropics or alt.sdn (smart drugs &
>> nutrients)? I have noticed some posts in sci.med.nutrition regarding
>>"smart nutrients." We may lower that groups burden as well.
>

I was wondering if a group called 'sci.pharmacology' would be relevant.
This would be used for a more formal discussion about pharmacological
issues (pharmacodynamics, neuropharmacology, etc.)

Just an informal proposal (I don't know anything about the net.politics
for adding a newsgroup, etc.)

"""

dict_list =[dict1, dict2]
data ={}
data["values"] = dict_list
input_data_json_str = json.dumps(data)
print (input_data_json_str)
prediction = web_service.score(input_data_json_str)
prediction

```

```
{"values": [{"recordId": "a1", "data": {"text": "\nI'd be interested in a copy of this code if you run across it.\n(Mail to the author bounced)\n / hpldsla:comp.graphics / email-address-removed / 12:53 am May 13,\n 1993 /\n I fooled around with this problem a few years ago, and implemented a\n simple method that ran on a PC.\n was very simple - about 40 or 50 lines of code.\n . .\n Somewhere I still have it\n and could dig it out if there was interest.\n"}}, {"recordId": "b2", "data": {"text": "\n>Could the people discussing recreational drugs such as mj, lsd, mdma, etc.,\n>take their discussions to alt.drugs? Their discussions will receive greatest\n>contribution and readership there. The people interested in strictly\n>"smart drugs"\n(i.e. Nootropics) should post to this group. The two groups\n>(alt.drugs & alt.psychactives) have been used interchangably lately.\n>I do think that alt.psychactives is a deceiving name. alt.psychactives\nis supposedly the \"smart drug\" newsgroup according to newsgroup lists on\nthe Usenet. Should we establish an alt.nootropics or alt.sdn (smart drugs &\n>nutrients)? I have noticed some posts in sci.med.nutrition regarding\n>"smart nutrients."\nWe may lower that groups burden as well.\n>\nI was wondering if a group called 'sci.pharmacology'\nwould be relevant.\nThis would be used for a more formal discussion about pharmacological\nissues (pharmacodynamics, neuropharmacology, etc.)\n\nJust an informal proposal (I don't know anything about the net.politics\nfor adding a newsgroup, etc.)\n\n"}]}]
```

F1 2018-05-02 00:10:58,272 INFO Web service scored.

```
'{"values": [{"recordId": "b2", "data": {"text": "\n>Could the people discussing recreational drugs such as mj, lsd, mdma, etc.,\n>take their discussions to alt.drugs? Their discussions will receive greatest\n>contribution and readership there. The people interested in strictly\n>"smart drugs"\n(i.e. Nootropics) should post to this group. The two groups\n>(alt.drugs & alt.psychactives) have been used interchangably lately.\n>I do think that alt.psychactives is a deceiving name. alt.psychactives\nis supposedly the \"smart drug\" newsgroup according to newsgroup lists on\nthe Usenet. Should we establish an alt.nootropics or alt.sdn (smart drugs &\n>nutrients)? I have noticed some posts in sci.med.nutrition regarding\n>"smart nutrients."\nWe may lower that groups burden as well.\n>\nI was wondering if a group called 'sci.pharmacology'\nwould be relevant.\nThis would be used for a more formal discussion about pharmacological\nissues (pharmacodynamics, neuropharmacology, etc.)\n\nJust an informal proposal (I don't know anything about the net.politics\nfor adding a newsgroup, etc.)\n\n"}, {"recordId": "a1", "data": {"text": "\nI'd be interested in a copy of this code if you run across it.\n(Mail to the author bounced)\n / hpldsla:comp.graphics / email-address-removed / 12:53 am May 13,\n 1993 /\n I fooled around with this problem a few years ago, and implemented a\n simple method that ran on a PC.\n was very simple - about 40 or 50 lines of code.\n . .\n Somewhere I still have it\n and could dig it out if there was interest.\n"}, "class": 13}], {"recordId": "a1", "data": {"text": "\nI'd be interested in a copy of this code if you run across it.\n(Mail to the author bounced)\n / hpldsla:comp.graphics / email-address-removed / 12:53 am May 13,\n 1993 /\n I fooled around with this problem a few years ago, and implemented a\n simple method that ran on a PC.\n was very simple - about 40 or 50 lines of code.\n . .\n Somewhere I still have it\n and could dig it out if there was interest.\n"}, "class": 13}]}'
```

Next steps

Learn more about Azure Machine Learning Package for Text Analytics in these articles:

- Read the [package overview](#).
- Explore the [reference documentation](#) for this package.
- Learn about [other Python packages for Azure Machine Learning](#).

Azure Machine Learning frequently asked questions

11/26/2018 • 10 minutes to read • [Edit Online](#)

NOTE

This article is deprecated. Support for earlier versions of this service will end incrementally. [View the support timeline](#). Start using the latest version with this [quickstart](#).

Azure Machine Learning is a fully managed Azure service that allows you to create, test, manage, and deploy machine learning and AI models. Our services and downloadable application offer a code-first approach that leverages the cloud, on-premises, and edge to provide the train, deploy, manage, and monitor models with power, speed, and flexibility. Alternatively, Azure Machine Learning Studio offers a browser-based, visual drag-and-drop authoring environment where no coding is required.

General product questions

What other Azure services are required?

Azure Blob Storage and Azure Container Registry are used by Azure Machine Learning. In addition, you will need to provision compute resources such as a Data Science VM or HDInsight cluster. Compute and hosting are also required when deploying your web services, such as [Azure Container Service](#).

How does Azure Machine Learning relate to Microsoft Machine Learning Services in SQL Server 2017?

Machine Learning Services in SQL Server 2017 is an extensible, scalable platform for integrating machine learning tasks into database workflows. It is a perfect fit for scenarios where an on-premises solution is required, for example where data movement is expensive or untenable. In contrast, cloud or hybrid workloads are a great fit for our new Azure services.

How does Azure Machine Learning relate to Microsoft Machine Learning for Spark?

MMLSpark provides deep learning and data science tools for Apache Spark, with emphasis on productivity, ease of experimentation and state-of-the-art algorithms. MMLSpark offers integration of Spark Machine Learning pipelines with the Microsoft Cognitive Toolkit and OpenCV. You can create powerful, highly scalable predictive, and analytical models for image and text data. MMLSpark is available under an open-source license and is included in AML Workbench as a set of consumable models and algorithms. For more information on MMLSpark, visit our product documentation.

Which versions of Spark are supported by the new tools and services?

Workbench currently includes and supports MMLSpark version 0.8, which is compatible with Apache Spark 2.1. You also have an option to use GPU-enabled Docker image of MMLSpark 0.8 on Linux virtual machines.

Experimentation Service

What is the Azure Machine Learning Experimentation Service?

The Experimentation Service is a managed Azure service that takes machine learning experimentation to the next level. Experiments can be built locally or in the cloud. Rapidly prototype on a desktop, then scale to virtual machines or Spark clusters. Azure VMs with the latest GPU technology allow you to engage in deep learning quickly and effectively. We've also included deep integration with Git so you can plug easily into existing workflows for code tracking, configuration, and collaboration.

How will I be charged for the Experimentation Service?

The first two users associated with your Azure Machine Learning Experimentation Service are free. Additional users will be charged at the Public Preview rate of \$50 / month. For more information on pricing and billing, visit our Pricing page.

Will I be charged based on how many experiments I run?

No, the Experimentation Service allows as many experiments as you need and charges only based on the number of users. Experimentation compute resources are charged separately. We encourage you to perform multiple experiments so you can find the best fitting model for your solution.

What specific kinds of compute and storage resources can I use?

The Experimentation service can execute your experiments on local machines (direct or Docker-based), [Azure Virtual Machines](#), and [HDInsight](#). The service also accesses an [Azure Storage](#) account for storing execution outputs and can leverage a [Visual Studio Team Service](#) account for version-control and Git storage. Note that you will be billed independently for any consumed compute and storage resources, based upon their individual pricing.

Model Management

What is Azure Machine Learning Model Management?

Azure Machine Learning Model Management is a managed Azure service that allows data scientists and dev-ops teams to deploy predictive models reliably into a wide variety of environments. Git repositories and Docker containers provide traceability and repeatability. Models can be deployed reliably in the cloud, on-premises, or edge. Once in production, you can manage model performance, then proactively retrain if performance degrades. You can deploy models on local machines, to [Azure VMs](#), Spark on [HDInsight](#) or Kubernetes-orchestrated [Azure Container Service](#) clusters.

What is a "model"?

A model is the output of an experimentation run that has been promoted for model management. A model that is registered in the hosting account is counted against your plan, including models updated through retraining or version iteration.

What is a "managed model"?

A model is the output of a training process and is the application of a machine learning algorithm to training data. Model Management enables you to deploy models as web services, manage various versions of your models, and monitor their performance and metrics. "Managed" models have been registered with an Azure Machine Learning Model Management account. As an example, consider a scenario where you are trying to forecast sales. During the experimentation phase, you generate many models by using different data sets or algorithms. You have generated four models with varying accuracies but choose to register only the model with the highest accuracy. The model that is registered becomes your first managed model.

What is a "deployment"?

Model Management allows you to deploy models as packaged web service containers in Azure. These web services can be invoked using REST APIs. Each web service is counted as a single deployment, and the total number of active deployments are counted towards your plan. Using the sales forecasting example, when you deploy your best performing model, your plan is incremented by one deployment. If you then retrain and deploy another version, you have two deployments. If you determine that the newer model is better, and delete the original, your deployment count is decremented by one.

What specific compute resources are available for my deployments?

Model Management can run your deployments as Docker containers registered to [Azure Container Service](#), as

[Azure Virtual Machines](#), or on local machines. Additional deployment targets will be added shortly. Note that you will be billed independently for any consumed compute resources, based upon their individual pricing.

Can I use the Azure Machine Learning Model Management to deploy models built using tools other than the Experimentation Service?

You get the best experience when you deploy models created using the Experimentation Service. However, you can deploy models built using other frameworks and tools. We support a variety of models including MMLSpark, TensorFlow, Microsoft Cognitive Toolkit, scikit-learn, Keras, etc.

Can I use my own Azure resources?

Yes, the Experimentation Service and Model Management work in conjunction with multiple Azure data stores, compute workloads, and other services. Refer to our technical documentation for further details on required Azure services.

Do you support both on-premises and cloud deployment scenarios?

Yes. We support on-premises and cloud deployment scenarios via Docker containers. Local execution targets include: single-node Docker deployments, [Microsoft SQL Server with ML Services](#), Hadoop, or Spark. We also support cloud deployments via Docker, including: clustered deployments via Azure Container Service and Kubernetes, HDInsight, or Spark clusters. Edge scenarios are supported via Docker containers and Azure IOT Edge.

Can I run a Docker image that was created using the Azure Machine Learning CLI on another host?

Yes. You can use the image as a web service on any docker host as long as host has sufficient compute resources for hosting the docker image.

Do you support retraining of deployed models?

Yes, you can deploy multiple versions of the same model. Model Management will support service updates for all updated models and images.

Workbench

What is the Azure Machine Learning Workbench?

The Azure Machine Learning Workbench is a companion application built for professional data scientists. Available for Windows and Mac, the Machine Learning Workbench provides overview, management, and control for machine learning solutions. The Machine Learning Workbench includes access to cutting edge AI frameworks from both Microsoft and the open source community. We've included the most popular data science toolkits, including TensorFlow, Microsoft Cognitive Toolkit, Spark ML, scikit-learn and more. We've also enabled integration with popular data science IDEs such as Jupyter notebooks, PyCharm, and Visual Studio Code. The Machine Learning Workbench has built-in data preparation capabilities to rapidly sample, understand, and prepare data, whether structured or unstructured. Our new data preparation tool, called [PROSE](#), is built on cutting-edge technology from Microsoft Research.

Is Workbench an IDE?

No. The Machine Learning Workbench has been designed as a companion to popular IDEs such as Jupyter Notebooks, Visual Studio Code, and PyCharm but it is not a fully functional IDE. The Machine Learning Workbench offers some basic text editing capabilities, but debugging, intellisense and other commonly used IDE capabilities are not supported. We recommend that you use your favorite IDE for code development, editing and debugging. You may also wish to try [Visual Studio Code Tools for AI](#).

Is there a charge for using the Azure Machine Learning Workbench?

No. Azure Machine Learning Workbench is a free application. You can download it on as many machines, and for

as many users, as you need. In order to use the Azure Machine Learning Workbench, you must have an Experimentation account. .

Do you support command-line capabilities?

Yes, Azure Machine Learning offers a full CLI interface. The Machine Learning CLI is installed by default with the Azure Machine Learning Workbench. It is also provided as part of the Linux Data Science virtual machine on Azure and will be integrated into the [Azure CLI](#)

Can I use Jupyter Notebooks with Workbench?

Yes! You can run Jupyter notebooks in Workbench, with Workbench as the client hosting application, just as you would use a browser as a client.

Which Jupyter Notebook kernels are supported?

The current version of Jupyter included with Workbench launches a Python 3 kernel, and an additional kernel for each "runconfig" file in your aml_config folder. Supported configurations include:

- Local Python
- Python in local or remote Docker

Data formats and capabilities

Which file formats are currently supported for data ingestion in Workbench?

The data preparation tools in Workbench currently support ingestion from the following formats:

- Delimited files such as CSV, TSV, etc.
- Fixed width files
- Plain text files
- Excel (.xls/xlsx)
- JSON files
- Parquet files
- Custom files (scripts) If your solution requires data ingestion from additional sources, Python code can be used to...

Which data storage locations are currently supported?

For public preview, Workbench supports data ingestion from:

- Local hard drive or mapped network storage location
- Azure BLOB or Azure Storage (requires an Azure subscription)
- Azure SQL Server
- Microsoft SQL Server

What kinds of data wrangling, preparation, and transformations are available?

For public preview, Workbench supports "Derive Column by Example", "Split Column by Example", "Text Clustering", "Handle Missing Values" and many others. Workbench also supports data type conversion, data aggregation (COUNT, MEAN, VARIANCE, etc.), and complex data joins. For a full list of supported capabilities, visit our product documentation.

Are there any data size limits enforced by Azure Machine Learning Workbench, Experimentation, or Model Management?

No, the new services do not impose any data limitations. However, there are limitations introduced by the environment in which you are performing your data preparation, model training, experimentation, or deployment.

For example, if you are targeting a local environment for training, you are limited by the available space in your hard drive. Alternatively, if you are targeting HDInsight, you are limited by any associated size or compute restraints.

Algorithms and libraries

What algorithms are supported in Azure Machine Learning Workbench?

Our preview products and services include the best of the open source community. We support a wide range of algorithms and libraries including TensorFlow, scikit-learn, Apache Spark, and the Microsoft Cognitive Toolkit. The Azure Machine Learning Workbench also packages the [Microsoft revoscalepy](#) package.

How does Azure Machine Learning relate to the Microsoft Cognitive Toolkit?

The [Microsoft Cognitive Toolkit](#) is one of many frameworks supported by our new tools and services. The Cognitive Toolkit is a unified deep-learning toolkit that allows you to consume and combine popular machine learning models including Feed-Forward Deep Neural Networks, Convolutional Nets, Sequence-to-Sequence, and Recurrent Networks. For more information on Microsoft Cognitive Toolkit, visit our [product documentation](#).

Azure Machine Learning Workbench - Known Issues And Troubleshooting Guide

10/9/2018 • 11 minutes to read • [Edit Online](#)

This article helps you find and correct errors or failures encountered as a part of using the Azure Machine Learning Workbench application.

Find the Workbench build number

When communicating with the support team, it is important to include the build number of the Workbench app. On Windows, you can find out the build number by clicking on the **Help** menu and choose **About Azure ML Workbench**. On macOS, you can click on the **Azure ML Workbench** menu and choose **About Azure ML Workbench**.

Machine Learning MSDN Forum

We have an MSDN Forum that you can post questions. The product team monitors the forum actively. The forum URL is <https://aka.ms/aml-forum-service>.

Gather diagnostics information

Sometimes it can be helpful if you can provide diagnostic information when asking for help. Here is where the log files live:

Installer log

If you run into issue during installation, the installer log files are here:

```
# Windows:  
%TEMP%\amlinstaller\logs\*  
  
# macOS:  
/tmp/amlinstaller/logs/*
```

You can zip up the contents of these directories and send it to us for diagnostics.

Workbench desktop app log

If you have trouble logging in, or if the Workbench desktop crashes, you can find log files here:

```
# Windows  
%APPDATA%\AmlWorkbench  
  
# macOS  
~/Library/Application Support/AmlWorkbench
```

You can zip up the contents of these directories and send it to us for diagnostics.

Experiment execution log

If a particular script fails during submission from the desktop app, try to resubmit it through CLI using `az ml experiment submit` command. This should give you full error message in JSON format, and most importantly it contains an **operation ID** value. Send us the JSON file including the **operation ID** and we can help

diagnose.

If a particular script succeeds in submission but fails in execution, it should print out the **Run ID** to identify that particular run. You can package up the relevant log files using the following command:

```
# Create a ZIP file that contains all the diagnostics information
$ az ml experiment diagnostics -r <run_id> -t <target_name>
```

The `az ml experiment diagnostics` command generates a `diagnostics.zip` file in the project root folder. The ZIP package contains the entire project folder in the state at the time it was executed, plus logging information. Be sure to remove any sensitive information you don't want to include before sending us the diagnostics file.

Send us a frown (or a smile)

When you are working in Azure ML Workbench, you can also send us a frown (or a smile) by clicking on the smiley face icon at the lower left corner of the application shell. You can optionally choose to include your email address (so we can get back to you), and/or a screenshot of the current state.

Known service limits

- Max allowed project folder size: 25 MB.

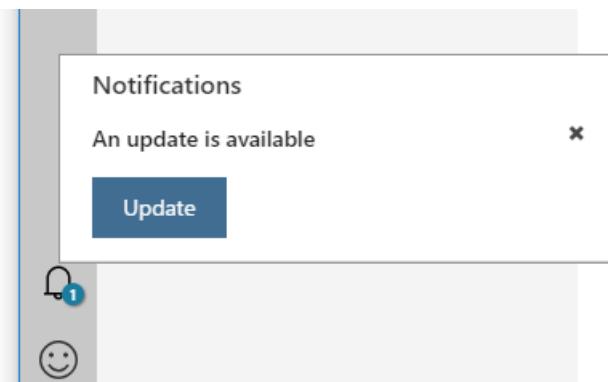
NOTE

This limit doesn't apply to `.git`, `docs` and `outputs` folders. These folder names are case-sensitive. If you are working with large files, refer to [Persisting Changes and Deal with Large Files](#).

- Max allowed experiment execution time: seven days
- Max size of tracked file in `outputs` folder after a run: 512 MB
 - This means if your script produces a file larger than 512 MB in the outputs folder, it is not collected there. If you are working with large files, refer to [Persisting Changes and Deal with Large Files](#).
- SSH keys are not supported when connecting to a remote machine or Spark cluster over SSH. Only username/password mode is currently supported.
- When using HDInsight cluster as compute target, it must use Azure blob as primary storage. Using Azure Data Lake Storage is not supported.
- Text clustering transforms are not supported on Mac.
- RevoScalePy library is only supported on Windows and Linux (in Docker containers). It is not supported on macOS.
- Jupyter Notebooks have a max size limit of 5 MB when opening them from the Workbench app. You can open large notebooks from CLI using 'az ml notebook start' command, and clean cell outputs to reduce the file size.

Can't update Workbench

When a new update is available, the Workbench app homepage displays a message informing you about the new update. You should see an update badge appearing on the lower left corner of the app on the bell icon. Click on the badge and follow the installer wizard to install the update.



If you don't see the notification, try to restart the app. If you still don't see the update notification after restart, there might be a few causes.

You are launching Workbench from a pinned shortcut on the task bar

You may have already installed the update. But your pinned shortcut is still pointing to the old bits on disk. You can verify this by browsing to the `%localappdata%\AmlWorkbench` folder and see if you have latest version installed there, and examine the property of the pinned shortcut to see where it is pointing to. If verified, simply remove the old shortcut, launch Workbench from Start menu, and optionally create a new pinned shortcut on the task bar.

You installed Workbench using the "install Azure ML Workbench" link on a Windows DSVM

Unfortunately there is no easy fix on this one. You have to perform the following steps to remove the installed bits, and download the latest installer to fresh-install the Workbench:

- remove the folder `C:\Users\<Username>\AppData\Local\amlworkbench`
- remove script `C:\dsvm\tools\setup\InstallAMLFromLocal.ps1`
- remove desktop shortcut that launches the above script
- download the installer <https://aka.ms/azureml-wb-msi> and reinstall.

Stuck at "Checking experimentation account" screen after logging in

After logging in, the Workbench app might get stuck on a blank screen with a message showing "Checking experimentation account" with a spinning wheel. To resolve this issue, take the following steps:

1. Shutdown the app
2. Delete the following file:

```
# on Windows  
%appdata%\AmlWorkbench\AmlWb.settings  
  
# on macOS  
~/Library/Application Support/AmlWorkbench/AmlWb.settings
```

3. Restart the app.

Can't delete Experimentation Account

You can use CLI to delete an Experimentation Account, but you must delete the child workspaces and the child projects within those child workspaces first. Otherwise, you see the error "Can not delete resource before nested resources are deleted."

```
# delete a project
$ az ml project delete -g <resource group name> -a <experimentation account name> -w <workspace name> -n <project name>

# delete a workspace
$ az ml workspace delete -g <resource group name> -a <experimentation account name> -n <workspace name>

# delete an experimentation account
$ az ml account experimentation delete -g <resource group name> -n <experimentation account name>
```

You can also delete the projects and workspaces from within the Workbench app.

Can't open file if project is in OneDrive

If you have Windows 10 Fall Creators Update, and your project is created in a local folder mapped to OneDrive, you might find that you cannot open any file in Workbench. This is due to a bug introduced by the Fall Creators Update that causes node.js code to fail in a OneDrive folder. The bug will be fixed soon by Windows update, but until then, please do not create projects in a OneDrive folder.

File name too long on Windows

If you use Workbench on Windows, you might run into the default maximum 260-character file name length limit, which could surface as a "system cannot find the path specified" error. You can modify a registry key setting to allow much longer file path name. Review [this article](#) for more details on how to set the *MAX_PATH* registry key.

Interrupt CLI execution output

If you kick off an experimentation run using `az ml experiment submit` or `az ml notebook start` and you'd like to interrupt the output:

- On Windows use Ctrl-Break key combination from the keyboard
- On macOS, use Ctrl-C.

Please note that this only interrupts the output stream in the CLI window. It does not actually stop a job that's being executed. If you want to cancel an ongoing job, use `az ml experiment cancel -r <run_id> -t <target name>` command.

On Windows computers with keyboards that do not have Break key, possible alternatives include Fn-B, Ctrl-Fn-B or Fn+Esc. Consult your hardware vendor's documentation for a specific key combination.

Docker error "read: connection refused"

When executing against a local Docker container, sometimes you might see the following error:

```
Get https://registry-1.docker.io/v2/:
dial tcp:
lookup registry-1.docker.io on [::1]:53: read udp [::1]:49385->[::1]:53:
read: connection refused
```

You can fix it by changing the Docker DNS Server from `automatic` to a fixed value of `8.8.8.8`.

Remove VM execution error "no tty present"

When executing against a Docker container on a remote Linux machine, you might encounter the following error message:

```
sudo: no tty present and no askpass program specified.
```

This can happen if you use Azure portal to change the root password of an Ubuntu Linux VM.

Azure Machine Learning Workbench requires password-less sudoers access to run on remote hosts. The simplest way to do that is to use `visudo` to edit the following file (you may create the file if it does not exist):

```
$ sudo visudo -f /etc/sudoers
```

IMPORTANT

It is important to edit the file with `visudo` and not another command. `visudo` automatically syntax checks all sudo config files, and failure to produce a syntactically correct sudoers file can lock you out of sudo.

Insert the following line at the end of the file:

```
username ALL=(ALL) NOPASSWD:ALL
```

Where `username` is the name of Azure Machine Learning Workbench will use to log in to your remote host.

The line must be placed after `#includedir "/etc/sudoers.d"`, otherwise it may be overridden by another rule.

If you have a more complicated sudo configuration, you may want to consult sudo documentation for Ubuntu available here: <https://help.ubuntu.com/community/Sudoers>

The above error can also happen if you are not using an Ubuntu-based Linux VM in Azure as an execution target. We only support Ubuntu-based Linux VM for remote execution.

VM disk is full

By default when you create a new Linux VM in Azure, you get a 30-GB disk for the operating system. Docker engine by default uses the same disk for pulling down images and running containers. This can fill up the OS disk and you see a "VM Disk is Full" error when it happens.

A quick fix is to remove all Docker images you no longer use. The following Docker command does just that. (Of course you have to SSH into the VM in order to execute the Docker command from a bash shell.)

```
$ docker system prune -a
```

You can also add a data disk and configure Docker engine to use the data disk for storing images. Here is [how to add a data disk](#). You can then [change where Docker stores images](#).

Or, you can expand the OS disk, and you don't have to touch Docker engine configuration. Here is [how you can expand the OS disk](#).

```

# Deallocate VM (stopping will not work)
$ az vm deallocate --resource-group myResourceGroup --name myVM

# Get VM's Disc Name
az disk list --resource-group myResourceGroup --query '[*].{Name:name,Gb:diskSizeGb,Tier:accountType}' --output table

# Update Disc Size using above name
$ az disk update --resource-group myResourceGroup --name myVMdisc --size-gb 250

# Start VM
$ az vm start --resource-group myResourceGroup --name myVM

```

Sharing C drive on Windows

If you are executing in a local Docker container on Windows, setting `sharedVolumes` to `true` in the `docker.compute` file under `aml_config` can improve execution performance. However, this requires you share C drive in the *Docker for Windows Tool*. If you are not able to share C drive, try the following tips:

- Check the sharing on C drive using file explorer
- Open network adapter settings and uninstall/reinstall "File and Printer Sharing for Microsoft Networks" for vEthernet
- Open docker settings and share C drive from within docker settings
- Changes to the Windows password affect the sharing. Open File explorer, reshare the C drive, and enter the new password.
- You might also encounter firewall issue when attempting to share your C drive with Docker. This [Stack Overflow post](#) can be helpful.
- When sharing C drive using domain credentials, the sharing might stop working on networks where the domain controller is not reachable (for example, home network, public wifi etc.). For more information, see [this post](#).

You can also avoid the sharing problem, at a small performance cost, by setting `sharedVolumne` to `false` in the `docker.compute` file.

Wipe clean Workbench installation

You generally don't need to do this. But in case you must wipe clean an installation, here are the steps:

- On Windows:
 - First make sure you use *Add or Remove Programs* applet in the *Control Panel* to remove the *Azure Machine Learning Workbench* application entry.
 - Then you can download and run either one of the following scripts:
 - [Windows command line script](#).
 - [Windows PowerShell script](#). (You may need to run `Set-ExecutionPolicy Unrestricted` in a privilege-elevated PowerShell window before you can run the script.)
- On macOS:
 - Just download and run the [macOS bash shell script](#).

Azure ML using a different python location than the Azure ML installed python environment

Due to a recent change in Azure Machine Learning Workbench, users may notice that local runs may not point to the python environment installed by the Azure ML Workbench anymore. This may happen if the user have another python environment installed on their computer and the "Python" path is set to point to that environment. In order

to use Azure ML Workbench installed Python environment, follow these steps:

- Go to local.compute file under your aml_config folder under your project root.
- Change the "pythonLocation" variable to point to the physical path of Azure ML workbench installed python environment. You can get this path in two ways:
 - Azure ML python location can be found at %localappdata%\AmlWorkbench\python\python.exe
 - you can open cmd from Azure ML Workbench, type python on command prompt, import sys.exe, run sys.executable and get the path from there.

Some useful Docker commands

Here are some useful Docker commands:

```
# display all running containers
$ docker ps

# display all containers (running or stopped)
$ docker ps -a

# display all images
$ docker images

# show Docker logs of a container
$ docker logs <container_id>

# create a new container and launch into a bash shell
$ docker run <image_id> /bin/bash

# launch into a bash shell on a running container
$ docker exec -it <container_id> /bin/bash

# stop an running container
$ docker stop <container_id>

# delete a container
$ docker rm <container_id>

# delete an image
$ docker rmi <image_id>

# delete all unussed Docker images
$ docker system prune -a
```