Operationalizing R with Azure Machine Learning

Contents

Overview

Overview

In this lab we will explore the functionality of the AzureML R package. You will learn how to use this package to upload and download datasets to and from AzureML, to interrogate experiments, to publish R functions as AzureML web services, and to run R data through existing web services and retrieve the output.

The AzureML package provides an interface to publish web services on Microsoft Azure Machine Learning (Azure ML) from your local R environment. The main functions in the package cover:

- **Workspace**: connect to and manage AzureML workspaces
- **Datasets**: upload and download datasets to and from AzureML workspaces
- **Publish**: define a custom function or train a model and publish it as an Azure Web Service
- **Consume**: use available web services from R in a variety of convenient formats

This lab focuses on small examples rather than trying to solve one particular use case. Therefore, please work through the examples and exercises.

Prerequisites

This lab is heavily focused on R and therefore a good understanding of R is required. In addition, it is helpful to have completed the following Azure Machine Learning labs:

- Introduction to Azure Machine Learning
- Deploying a Predictive Model with Azure Machine Learning
- Text Analytics with R and Azure Machine Learning

System requirements

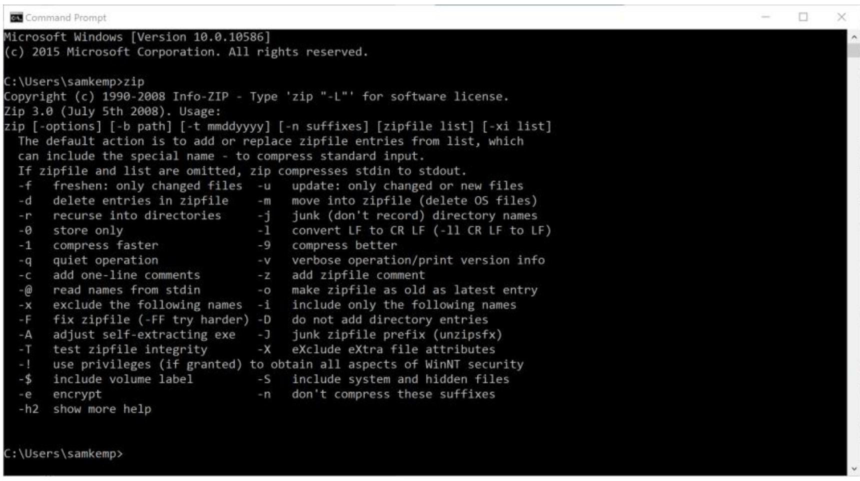
To publish web services, you need to have an external zip utility installed. This utility should be in the available in the path. See `?zip` (from the R console) for more details.

If you are using the Data Science Virtual Machine (DSVM) in Azure then this already has zip installed via RTools.

If you are **NOT** using the DSVM in Azure then you need to following this guidance:

On Windows, it's sufficient to install RTools. N.B. If you completed the *“Advanced Analytics Lab: Prerequisite activity”* then you have already installed RTools on your Data Science Virtual Machine.

Note: the utility should be called **zip**, since **zip()** looks for a file called zip in the path. Thus, **publishWebservice()** may fail, even if you have a program like 7-zip installed. To test if you have zip installed type **zip** at the command line prompt you should see the following output:



Installation

If you are NOT using the Data Science Virtual Machine in Azure then you will need to install the AzureML package copy-and-paste the following into the R console.

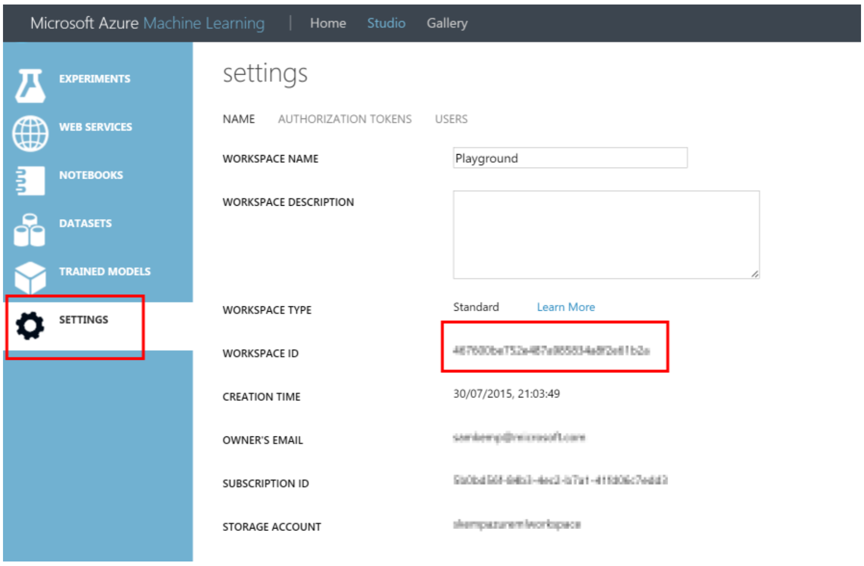
```
install.packages("AzureML")

library(AzureML)
```

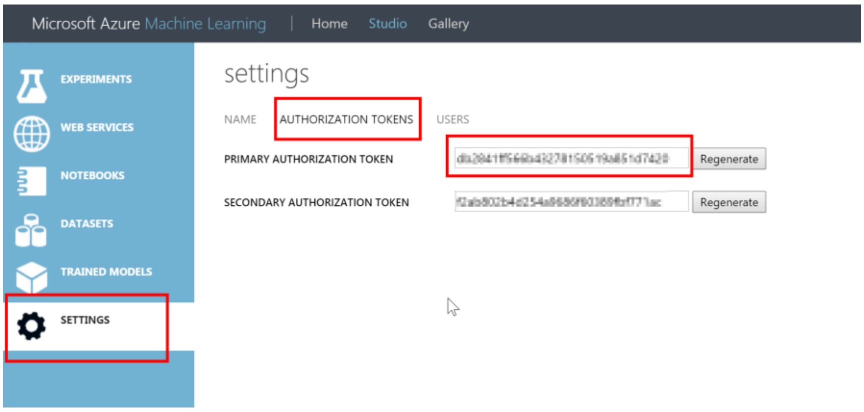
Getting Started: The workspace object

The package defines a Workspace class that represents an AzureML work space. Most of the functions in the package refer to a Workspace object directly or indirectly. Use the `workspace()` function to create Workspace objects, either by explicitly specifying an AzureML workspace ID and authorization token. Workspace objects are simply R environments that actively cache details about your AzureML sessions.

In order to publish a web service to Azure Machine Learning you will need your Workspace ID and Authorization Key. Navigate to the Azure Machine Learning Studio - https://studio.azureml.net/ - You can find your Workspace ID by going to Azure ML Studio > **Settings**:



The Authorization token can be found by going to **Authorization Tokens** > **Primary Key**.



Copy-and-paste the workspace ID and Authorization token into the following R string variables.

Note: This code snippet assumes your workspace is located in the West Europe data center. If your workspace is located in a different data center then you will need to change the `api_endpoint` and `management_endpoint` parameters accordingly. If you are using the Free Azure ML account, then you do only need to specify `id` and `auth`.

```
ws <- workspace(id = "WORKSPACE ID",
                auth = "AUTH KEY",
                api_endpoint =
"https://europewest.studio.azureml.net",
                management_endpoint =
"https://europewest.management.azureml.net")
```

Examining the workspace

The `datasets()`, `experiments()`, and `services()` functions return data frames that contain information about those objects available in the workspace.

The package caches R data frame objects describing available datasets, experiments and services in the workspace environment. That cache can be refreshed at any time with the `refresh()` function. The data frame objects make it relatively easy to sort and filter the datasets, experiments, and services in arbitrary ways. The functions also include filtering options for specific and common filters, like looking up something by name.

Use the `download.datasets()` and `upload.dataset()` functions to download or upload data between R and your Azure workspace. The `download.intermediate.dataset()` function can download ephemeral data from a port in an experiment that is not explicitly stored in your Azure workspace.

Use `delete.datasets()` to remove and delete datasets from the workspace.

The `endpoints()` function describes Azure web service endpoints, and works with supporting help functions like `endpointHelp()`.

The `publishWebService()` function publishes a custom R function as an AzureML web service, available for use by any client. The `updateWebService()` and `deleteWebService()` update or delete existing web services, respectively.

Use the `consume()` function to evaluate an Azure ML web service with new data uploaded to AzureML from your R environment.

Datasets

AzureML datasets correspond more or less to R data frames. The AzureML package defines four basic dataset operations: list, upload, download, and delete.

To view a list of the available datasets in the workspace you can use the `datasets()` function:

```
head(datasets(ws))      # Or, equivalently: head(ws$datasets)
```

```
> datasets(ws)
                                              Name         DataTypeId     Size ...
1                Telco Churn Training Data - Copy (2)            Dataset   456315 ...
2                           Telco Churn Training Data            Dataset   456315 ...
3                Telco Churn Training Data - Copy (3)            Dataset   456315 ...
4                Telco Churn Training Data - Copy (4)            Dataset   456315 ...
5                    TelcoCustomerChurnTrainingSample         GenericCSV  3110940 ...
6                TelcoCustomerChurnTrainingSample.csv         GenericCSV  3110940 ...
7                               text.preprocessing.zip                Zip     2782 ...
8                                fraudTemplateUtil.zip                Zip     3471 ...
9            Sample Named Entity Recognition Articles GenericTSVNoHeader      236 ...
10   testDataSource_691751f888654058b0242e521d1b8a05         GenericTSV       25 ...
11   MetaAnalytics.Test.GlobalDataset.IntegerCSVFile         GenericCSV       27 ...
12   MetaAnalytics.Test.GlobalDataset.IntegerTSVFile         GenericTSV       26 ...
13   testDataSource_be1d4a6ab8fa4444acad3d40be3f3f76         GenericTSV       25 ...
14                                Breast cancer data               ARFF    15170 ...
15                                 Forest fires data               ARFF    26285 ...
16                               Iris Two Class Data               ARFF     2004 ...
17 Adult Census Income Binary Classification dataset         GenericCSV  4007034 ...
18                Steel Annealing multi-class dataset         GenericCSV    81431 ...
19                        Automobile price data (Raw)         GenericCSV    26420 ...
20                    MPG data for various automobiles         GenericCSV    17867 ...
21                               Blood donation data         GenericCSV    12769 ...
```

Because the package **caches** objects available in the workspace environment, it is also possible to use the following syntax to get a list of datasets:

```
ws$datasets
```

The list of datasets is presented as an R data frame with class `Datasets`. Its print method shows a summary of the datasets, along with all of the available variables. Use any normal R data frame operation to manipulate the datasets. For example, to see the “Owner” value of each dataset:

```
head(ws$datasets$Owner, n=20)
```

Downloading datasets

The next example illustrates downloading a specific dataset named “Airport Codes Dataset” from AzureML to your R session. This dataset is presented by AzureML as a “Generic CSV” dataset, and will be parsed by R’s `read.table()` function.Other formats are parsed by an appropriate parser, for example `read.arff()`.) The example illustrates passing additional arguments to the `read.table()` function used to parse the data from AzureML in this case.

```
airports <- download.datasets(ws, name = "Airport Codes Dataset",
quote="\"")
head(airports)
```

```
> airports <- download.datasets(ws, name = "Airport Codes Dataset", quote="\"")
> head(airports)
  airport_id        city state                               name
1      10165 Adak Island    AK                               Adak
2      10299   Anchorage    AK  Ted Stevens Anchorage International
3      10304       Aniak    AK                      Aniak Airport
4      10754      Barrow    AK       Wiley Post/Will Rogers Memorial
5      10551      Bethel    AK                     Bethel Airport
6      10926     Cordova    AK                Merle K Mudhole Smith
> 
```

Note: You can use `download.datasets()` to download more than one dataset as a time, returning the results in a list of data frames.

Uploading datasets

Use the `upload.dataset()` function to upload R data frames to AzureML.

```
upload.dataset(airquality, ws, name = "Air quality")
```

```
> upload.dataset(airquality, ws, name = "Air quality")
         Name DataTypeId Size ...
1 Air quality GenericTSV 2901 ...
----------------------------------------------
AzureML datasets data.frame variables include:
 [1] "VisualizeEndPoint"     "SchemaEndPoint"        "SchemaStatus"          "Id"                    "DataTypeId"
 [6] "Name"                  "Description"           "FamilyId"              "ResourceUploadId"      "SourceOrigin"
[11] "Size"                  "CreatedDate"           "Owner"                 "ExperimentId"          "ClientVersion"
[16] "PromotedFrom"          "UploadedFromFilename"  "ServiceVersion"        "IsLatest"              "Category"
[21] "DownloadLocation"      "IsDeprecated"          "Culture"               "Batch"                 "CreatedDateTicks"
> 
```

We can see what happened by issuing the following R code:

```
head(download.datasets(ws, name = "Air quality"))
```

Deleting datasets

Delete one or more AzureML datasets with `delete.datasets()`:

```
delete.datasets(ws, name="Air quality")
```

Experiments

Use the `experiments()` function or simply use the `ws$experiments` data frame object directly to list details about experiments in your AzureML workspace. The `experiments()` function optionally filters experiments by ownership.

```
e <- experiments(ws)
head(e)
```

```
> e <- experiments(ws)
> head(e)
                                  Description        CreationTime ...
1                AA Lab ADF-AML [Predictive Exp.] 2015-11-05 21:51:44 ...
2                         Time Series Forecasting 2015-10-15 16:27:31 ...
3               CATelcoCustomerChurnTrainingSample 2015-10-26 10:33:13 ...
4 Binary Classification: Twitter sentiment analysi 2015-10-22 17:00:11 ...
5                     CATelcoCustomerChurnScoring 2015-11-05 15:55:32 ...
6 Pass Summit 2015 Data Storytelling with R AzureM 2015-10-31 20:58:46 ...
-------------------------------------------------
AzureML experiments data.frame variables include:
 [1] "ExperimentId"                      "Description"                     "Etag"
 [4] "Creator"                           "IsArchived"                      "JobId"
 [7] "VersionId"                         "RunId"                           "OriginalExperimentDocumentationLink"
[10] "Summary"                           "Category"                        "Tags"
[13] "StatusCode"                        "StatusDetail"                    "CreationTime"
[16] "StartTime"                         "EndTime"                         "Metadata"
> 
```

The `ws$experiments` object is just an R data frame with class Experiments. Its print method shows a summary of the available experiments, but it can otherwise be manipulated like a normal R data frame.

The list of experiments in your workspace is cached in the workspace environment. Use the `refresh()` function to explicitly update the cache at any time, for example:

```
refresh(ws, "experiments")
```

Web Services

The AzureML package helps you to publish R functions (which can contain predictive model objects to score against) as AzureML web services that can be consumed anywhere. You can also use the AzureML package to run R data through an existing web service and collect the output.

The ability to easily publish an R function as a web-service in Azure Machine Learning allows us to:

- Harness other Cortana Analytics components to produce an end-to-end production grade analytics/predictive system in the cloud. For example, we can leverage Azure Data Factory to do ETL and aggregation of raw data and then score that data using an R model.

- Use the Excel AzureML add-in to query an R web service and display the results in Excel. We are therefore, leveraging R's excellent quantitative analytics capability inside Excel.

- Develop web-sites that utilize machine learning algorithms to enhance the customer experience e.g. recommendation algorithms, churn analysis, tailored advertising.

- Centralize R code into a single repository for other users to consume.

The `publishWebService()` publishes an R function as an AzureML web service. Consider this simple example R function:

```
add <- function(x, y) {
  x + y
}
```

Use the function `publishWebService()` to publish the function as a service named "aalab -silly":

```
api <- publishWebService(
  ws,
  fun = add,
  name = "aalab-silly",
  inputSchema = list(
    x = "numeric",
    y = "numeric"
  ),
  outputSchema = list(
    ans = "numeric"
  )
)

api
```

The example publishes a function of two scalar numeric arguments, returning a single numeric scalar output value. Note that we explicitly define the web service input and output schema in the example. See the examples below for more flexible ways of defining web services with functions of data frames.

The result of `publishWebService()` is an Endpoint object i.e. an R data frame with two elements: a list containing the details of the newly created web service, and a list of the endpoints of the web service. From here, you can pass the information on to another user, or use the information to use the web service from R.

The **web service created is identical to a web service published through the Azure Machine Learning Studio**. From the response, you can get the Web Service's URL, API Key and Help Page URL, as shown above. The first two are needed to make calls to the web service. The latter has the sample code, sample request and other information for consuming the API from client apps such as mobile and web applications.

The new web service will show up on the 'Web Services' tab of the Studio interface. Go ahead and test the service in Azure Machine Learning Studio.

Note that AzureML allows multiple services to have the same name.

Updating Web Services

Once published, you can update a web service using the `updateWebService()` or `publishWebService()` functions. The `updateWebService()` function is just an alias for `publishWebService()`, except that the argument serviceId is compulsory.

For example, to change the "aalab-silly" service to subtract two numbers instead of adding them:

```
api <- updateWebService(
  ws,
  fun = function(x, y) x - y,
  inputSchema = list(
    x = "numeric",
    y = "numeric"
  ),
  outputSchema = list(
    ans = "numeric"
  ),
  serviceId = api$WebServiceId   # <<-- Required to update!
)
```

Discovering Web Services

Use the `services()` function to list in detail all of the available services in your AzureML workspace, or filter by web service name as shown below:

```
(webservices <- services(ws, name = "aalab-silly"))
```

Given a service, use the `endpoints()` function to list the AzureML service endpoints for the service:

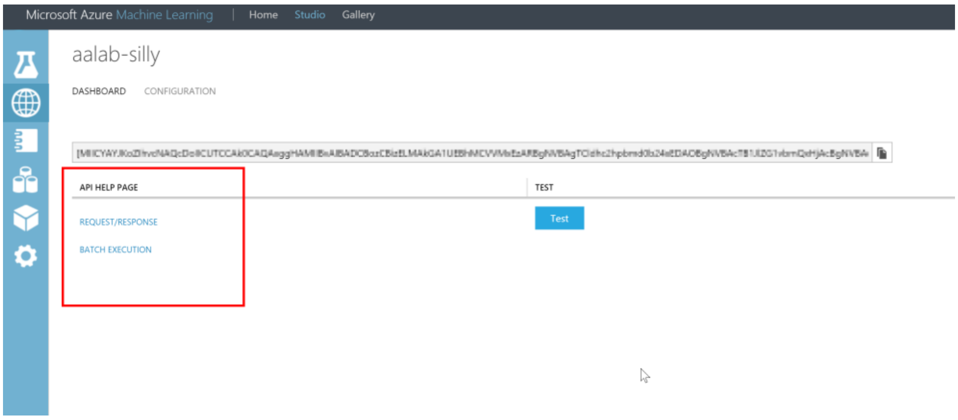
```
ep <- endpoints(ws, webservices[1, ])
```

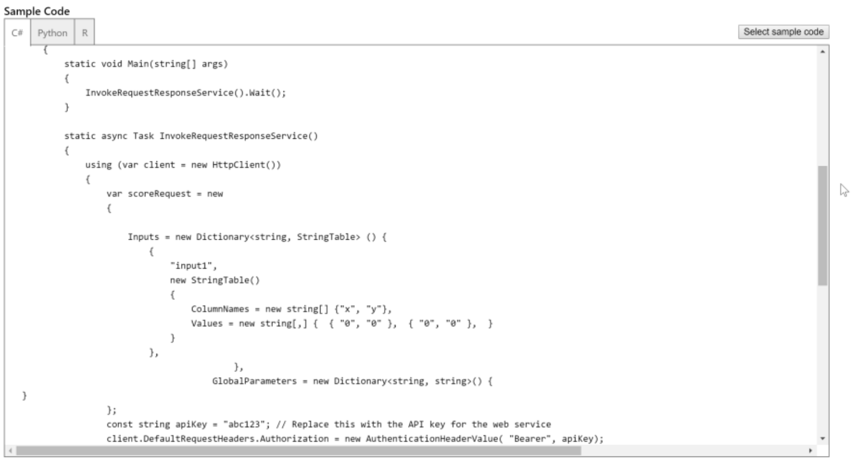
Consuming Web Services

Using the `consume()` function to send data to your newly published web service API for scoring.

```
df <- data.frame(
  x = 1:5,
  y = 6:10
)
s <- services(ws, name = "aalab-silly")
s <- tail(s, 1) # use the last published function, in case of duplica
te function names
ep <- endpoints(ws, s)
consume(ep, df)
```

Alternatively, we may want to build an application (Python or C#) – possibly web-based – that consumes the web service. The Help Page URL provides some sample code to get you going:





Exercise

Take the C# code sample from the Help page and build the application in Visual Studio.

Hints: Follow the guidelines at the top of the sample code i.e.

```
 // This code requires the Nuget package
Microsoft.AspNet.WebApi.Client to be installed.
// Instructions for doing this in Visual Studio:
// Tools -> Nuget Package Manager -> Package Manager
Console
// Install-Package Microsoft.AspNet.WebApi.Client
```

Update the API key in the C# code – you will find this in the using the `endpoints()` function in the in R.

Deleting Web Services

Use `deleteWebservice()` to remove a web service endpoint that you no longer need or want:

```
deleteWebService(ws, name = "aalab-silly")
```

Publishing Predictive Models

The simplest and perhaps most useful way to define a web service uses functions that take a single data frame argument and return a vector or data frame of results. The next example trains a generalized boosted regression model using the gbm package, publishes the model as a web service with name “aalab-gbm”, and runs example data through the model for prediction using the `consume()` function. For this example, we use the “Boston Housing Data”, which is included in the MASS package. This data contains housing values in the suburbs of Boston along with potential predictively useful information.

```
install.packages("gbm")
library(MASS)
library(gbm)

test <- Boston[1:5, 1:13]

set.seed(123)
gbm1 <- gbm(medv ~ .,
            distribution = "gaussian",
            n.trees = 5000,
            interaction.depth = 8,
            n.minobsinnode = 1,
            shrinkage = 0.01,
            cv.folds = 5,
            data = Boston)
best.iter <- gbm.perf(gbm1, method="cv", plot=FALSE)

mypredict <- function(newdata)
{
  require(gbm)
  predict(gbm1, newdata, best.iter)
}

# Example use of the prediction function
print(mypredict(test))
```

we publish the web service using the following (notice that we don’t need to explicitly define
the `inputSchema` or `outputSchema` arguments when working with functions that use data frame I/O):

```
ep <- publishWebService(ws = ws, fun = mypredict, name = "aalab-gbm",
inputSchema = test)
```

the web service can then be consumed using:

```
print(consume(ep, test))
```

When using data frames as inputs to the Web Service function you cannot specify additional parameters in the function i.e. function input must be either:

1. named scalar arguments with names and types specified in `inputSchema`

2. One or more lists of named scalar values

3. A single data frame when `data.frame=TRUE` is specified; either explicitly specify the column names and types in `inputSchema` or provide an example input data frame as `inputSchema`

Additional Tips

Try to use the data frame I/O interface as illustrated in the last example above. It's simpler and more robust than using functions of scalars or lists and exhibits faster execution for large data sets.

Use require in your function to explicitly load required packages.

The `publishWebService()` function uses `codetools` to bundle objects required by your function following R lexical scoping rules. The previous example, for instance, uses the `best.iter` and `gbm1` variables inside of the `mypredict()` function. `publishWebService()` identified that and included their definitions in the R environment in which the function is evaluated in AzureML. Fine-grained control over the export of variables is provided by the `publishWebService()` function in case you need it (see the help page for details).

Use the packages option of `publishWebService()` to explicitly bundle required packages and their dependencies (but not suggested dependencies) using miniCRAN. This lets you upload packages to AzureML that may not otherwise be available in that environment already, using the correct R version and platform used by AzureML.

As an example, let's say we want to operationalize an integer programming problem to solve the transport problem using the lpSolve package in R.

We create a function that uses the lp.transport() function to solve the problem for a given cost matrix:

```
install.packages("lpSolve")
library(lpSolve)
costs <- matrix (10000, 8, 5); costs[4,1] <- costs[-4,5] <- 0
costs[1,2] <- costs[2,3] <- costs[3,4] <- 7; costs[1,3] <- costs[2,4]
<- 7.7
costs[5,1] <- costs[7,3] <- 8; costs[1,4] <- 8.4; costs[6,2] <- 9
costs[8,4] <- 10; costs[4,2:4] <- c(.7, 1.4, 2.1)
costs <- data.frame(costs)

row.signs <- rep ("<", 8)
row.rhs <- c(200, 300, 350, 200, 100, 50, 100, 150)
col.signs <- rep (">", 5)
col.rhs <- c(250, 100, 400, 500, 200)


mylpSolver <- function(myCosts)
{
  return(data.frame(lp.transport (as.matrix(myCosts), "min",
row.signs, row.rhs, col.signs, col.rhs)$solution))
}

mylpSolver(costs)
```

We then publish the function as a web service and include lpSolve as a package to upload:

```
ep <- publishWebService(ws = ws, fun = mylpSolver, name = "aalab-lpso
lver",inputSchema = list(X1="numeric", X2="numeric",X3="numeric",X4="
numeric",X5="numeric"), outputSchema = list(X1="numeric", X2="numeric
",X3="numeric",X4="numeric",X5="numeric"),                          pac
kages = "lpSolve", data.frame = TRUE)
```

You will see that the publishWebService() function zips the lpSolve package into a miniCRAN repository and includes this in the web service, i.e.

```
> ep <- publishWebService(ws = ws, fun = mylpSolver, name = "aalab-lpsolver",inputSchema = list(X1="n
X2="numeric",X3="numeric",X4="numeric",X5="numeric"),
+                           outputSchema = list(X1="numeric", X2="numeric",X3="numeric",X4="numeric",X5
"),packages = "lpSolve", data.frame = TRUE)
Created new folder: C:/Users/samkemp/AppData/Local/Temp/2/Rtmp2VuGFB/dir8f841d1758f/packages/bin/wind
ib/3.1
trying URL 'https://mran.revolutionanalytics.com/snapshot/2015-07-01/bin/windows/contrib/3.1/lpSolve_
p'
Content type 'application/zip' length 676266 bytes (660 KB)
downloaded 660 KB

  adding: env.RData (stored 0%)
  adding: packages/ (stored 0%)
  adding: packages/bin/ (stored 0%)
  adding: packages/bin/windows/ (stored 0%)
  adding: packages/bin/windows/contrib/ (stored 0%)
  adding: packages/bin/windows/contrib/3.1/ (stored 0%)
  adding: packages/bin/windows/contrib/3.1/lpSolve_5.6.11.zip (deflated 0%)
  adding: packages/bin/windows/contrib/3.1/PACKAGES (stored 0%)
  adding: packages/bin/windows/contrib/3.1/PACKAGES.gz (stored 0%)
> print(consume(ep, costs))
```

To consume the web-service we use:

```
print(consume(ep, costs))
```

Be aware that the version of R running in AzureML may not be the same as the version of R that you are running locally. That means that some packages might not be available, or sometimes package behavior in the AzureML version of R might be different than what you observe locally. This is generally more of an issue for cutting-edge packages.

JSON is used to transfer data between your local R environment and the R services running in AzureML–numeric values experience a change of base, which can lead to a small loss of precision in some circumstances. If you really, really need to move binary objects between your local R session and the AzureML R service you might try base64 encoding the data, for example.

Exercise

1. Download the “Bike Rental UCI dataset” into a data frame using the `download.datasets()` function. This dataset contains the number of bikes rented at hourly intervals and also contains predictively useful information regarding the weather and whether or not it was a holiday.

   The objective of this exercise is to create a model that can forecast the hourly demand for the rental bikes.

2. Create a linear model where the response variable is the **cnt** variable (number of bikes rented in an hour)

3. Create a function that uses the model to produce a prediction.

4. Publish the model as a web service.

5. Test the model using the `consume()` function.

6. Update the web service with a model that uses a random forest (tip: you will need to install-and-load the `randomForest` package).

Terms of Use

By using this Hands-on Lab, you agree to the following terms:

The technology/functionality described in this Hands-on Lab is provided by Microsoft Corporation in a “sandbox” testing environment for purposes of obtaining your feedback and to provide you with a learning experience. You may only use the Hands-on Lab to evaluate such technology features and functionality and provide feedback to Microsoft. You may not use it for any other purpose. You may not modify copy, distribute, transmit, display, perform, reproduce, publish, license, create derivative works from, transfer, or sell this Hands-on Lab or any portion thereof.

COPYING OR REPRODUCTION OF THE HANDS-ON LAB (OR ANY PORTION OF IT) TO ANY OTHER SERVER OR LOCATION FOR FURTHER REPRODUCTION OR REDISTRIBUTION IS EXPRESSLY PROHIBITED.

THIS HANDS-ON LAB PROVIDES CERTAIN SOFTWARE TECHNOLOGY/PRODUCT FEATURES AND FUNCTIONALITY, INCLUDING POTENTIAL NEW FEATURES AND CONCEPTS, IN A SIMULATED ENVIRONMENT WITHOUT COMPLEX SET-UP OR INSTALLATION FOR THE PURPOSE DESCRIBED ABOVE. THE TECHNOLOGY/CONCEPTS REPRESENTED IN THIS HANDS-ON LAB MAY NOT REPRESENT FULL FEATURE FUNCTIONALITY AND MAY NOT WORK THE WAY A FINAL VERSION MAY WORK. WE ALSO MAY NOT RELEASE A FINAL VERSION OF SUCH FEATURES OR CONCEPTS. YOUR EXPERIENCE WITH USING SUCH FEATURES AND FUNCITONALITY IN A PHYSICAL ENVIRONMENT MAY ALSO BE DIFFERENT.

FEEDBACK. If you give feedback about the technology features, functionality and/or concepts described in this Hands-on Lab to Microsoft, you give to Microsoft, without charge, the right to use, share and commercialize your feedback in any way and for any purpose. You also give to third parties, without charge, any patent rights needed for their products, technologies and services to use or interface with any specific parts of a Microsoft software or service that includes the feedback. You will not give feedback that is subject to a license that requires Microsoft to license its software or documentation to third parties because we include your feedback in them. These rights survive this agreement.

MICROSOFT CORPORATION HEREBY DISCLAIMS ALL WARRANTIES AND CONDITIONS WITH REGARD TO THE HANDS-ON LAB, INCLUDING ALL WARRANTIES AND CONDITIONS OF MERCHANTABILITY, WHETHER EXPRESS, IMPLIED OR STATUTORY, FITNESS FOR A PARTICULAR PURPOSE, TITLE AND NON-INFRINGEMENT. MICROSOFT DOES NOT MAKE ANY ASSURANCES OR REPRESENTATIONS WITH REGARD TO THE ACCURACY OF THE RESULTS, OUTPUT THAT DERIVES FROM USE OF THE VIRTUAL LAB, OR SUITABILITY OF THE INFORMATION CONTAINED IN THE VIRTUAL LAB FOR ANY PURPOSE.

DISCLAIMER

This lab contains only a portion of the features and enhancements in Microsoft Azure. Some of the features might change in future releases of the product.