H2O Developer Cookbook

Preface

H2O is an in-memory engine for predictive analytics and machine learning.

Discuss the differences between ValueArray and Fluid Vector here.

1 Getting Started from an IDE

1.1 Cloning H2O from Github

1.2 Downloading Java

1.3 Importing the H2O project into Eclipse

1.4 Compiling H2O in Eclipse

1.5 Starting H2O in Eclipse

1.6 Importing the H2O project into IntelliJ IDEA

1.7 Compiling H2O in IntelliJ IDEA

1.8 Starting H2O in IntelliJ IDEA

2 Getting Started from the Command Line

2.1 Compiling H2O from the command line

2.2 Running H2O from the command line

2.3 Running a multi-node H2O cluster from the command line

3 Reading Data

3.1 Reading a CSV file from local disk

3.2 Reading a CSV file from HDFS

3.3 Reading a CSV file from S3

3.4 Reading a directory of CSV files from local disk

3.5 Reading a directory of CSV files from HDFS

3.6 Reading a directory of CSV files from S3

4 Introduction to Representation of Data

Data in H2O is referenced through a Frame. Frames are loosely analogous to Data Frames in R, although in H2O the actual data is stored in vectors (a Vec) rather than in the Frame itself. A Vec may be referenced by more than one Frame. Each Frame is composed of one or more Vecs. Each Vec is composed of one or more Chunks. Each Chunk is composed on one or more Elements.

4.1 Element

Some text here about Elements.

4.1.1 Double elements

4.1.2 Long elements

4.1.3 Enum (aka Categorical) elements

4.1.4 Missing (aka NA) values

4.1.5 Not-a-Number (aka NaN) values

4.1.6 Infinity values

4.2 Chunk

Some text here about Chunks

4.2.1 Reading a double element from a Chunk

4.2.2 Reading a long element from a Chunk

4.2.3 Updating a double element to a Chunk

4.2.4 Updating a long element to a Chunk

4.3 Vec

Some text here about Vecs

4.3.1 Reading an element from a Vec

4.3.2 Updating an element to a Vec

4.3.3 Testing whether a Vec of integers is an Enum (aka Categorical) or not

4.3.4 Printing the list of Domains of an Enum Vec (aka Levels of a Categorical Vec)

4.3.5 Updating a Vec element with an Enum value that has never been used before

4.3.6 Accessing Vec stats that are computed automatically (like min, max)

4.4 Frame

Some text here about Frames

4.4.1 Accessing a Vec from a Frame

4.4.2 Printing all column (aka Vec) names in a Frame

4.4.3 Adding an existing Vec to a Frame

4.4.4 Removing a Vec from a Frame

4.4.5 Creating a new Frame with a subset of Vecs from an existing Frame

4.4.6 Removing all Frame references to a Vec and reclaiming its memory

4.4.7 Creating a new double Vec from nothing and adding it to a Frame

4.4.8 Creating a new long Vec from nothing and adding it to a Frame

4.4.9 Creating a new Enum Vec from nothing and adding it to a Frame

5 Sharing Data Across Nodes

5.1 DKV (Distributed Key/Value Store)

DKV stands for Distributed Key/Value store. The DKV is the high-performance atomic distributed store that provides the clustering support for data in H2O.

5.1.1 Reading a value from the DKV

5.1.2 Writing a new value to the DKV

5.1.3 Updating a value in the DKV

5.1.4 Removing a value from the DKV

5.1.5 Writing multiple values to the DKV

5.1.6 Updating multiple values in the DKV

5.1.7 Removing multiple values from the DKV

5.2 UKV (User-level Key/Value Store)

The UKV is an abstraction on top of the DKV.

5.2.1 Adding compound objects to the UKV

5.2.2 Removing compound objects from the UKV

5.2.3 Looking at UKV objects using the Web UI

5.3 Vecs and Frames

5.3.1 Reading a value updated by a different node

5.4 Chunks and Data Parallelism

5.4.1 Observing how Chunk arrays are the basic unit of parallelism

5.4.2 Observing how Chunk arrays get sprayed across a Cluster for small data

5.4.3 Observing how Chunk arrays get sprayed across a Cluster for big data

6 Clusters

6.1 Knowing when a cluster is ready for use

6.2 Checking cluster health

7 Working with Data that exceeds the size of Memory

7.1 Managing the location of temporary files by setting ICE_ROOT

7.2 Reading a huge CSV file

8 Object Serialization

8.1 Freezable and Iced

8.1.1 Defining a new Iced object

8.1.2 Sending an Iced object across the network

9 Tasks

9.1 DRemoteTask

9.1.1 Running a piece of Java code on all H2O nodes

9.2 MRTask2

9.2.1 Writing a new MRTask2 task to sum the values of a column

9.2.2 Returning the value of a MRTask2 task as a member of an Iced object

9.2.3 Returning the value of a MRTask2 task by adding new Vecs (aka columns) to a Frame

10 Jobs

A Job is a major piece of work that gets added to the Jobs list and is visible in the Jobs Web UI page. A Job is generally reserved for things that produce an output you would be interested keeping around (a model, for example). Top-level algorithms like Random Forest are implemented as a Job.

10.1 Creating a new Job

10.2 Starting a Job

10.3 Monitoring a Job

11 Scala and H2O

11.1 Getting started with Shalala

11.2 Reading data in from Scala

11.3 Writing Scala scripts for H2O

11.4 Writing a new MRTask2 task in Scala

11.5 Calling Java MRTask2 tasks from Scala

11.6 Running Scala code in a multi-node H2O cluster

12 REST API

12.1 Creating a new REST API endpoint

13 Logging

13.1 Writing new log statements

13.2 Finding log files

13.3 Configuring a custom logging setup

14 Using H2O as a Dependency for a new Project

14.1 Compiling your project with H2O as a dependency

14.2 Registering a new REST API endpoint with H2O

14.3 Running an H2O cluster with a new REST endpoint from IntelliJ IDEA

14.4 Running an H2O cluster with a new REST endpoint from the command line

15 Embedding H2O

15.1 Running H2O inside a Hadoop map task

16 Testing

16.1 Running JUnit tests from the command line

16.2 Running JUnit tests from an IDE

16.3 Running one RUnit test from the command line

16.4 Running one RUnit test from RStudio

16.5 Running all the RUnit tests from the command line

16.6 Running one Python test from the command line

16.7 Running all the Python tests from the command line

16.8 Running H2O CRAN package examples from the command line

16.9 Adding a JUnit test

16.10 Adding an RUnit test

16.11 Adding a Python test

17 Common Pitfalls

17.1 Frequently hit assertions by new H2O developers

17.1.1 Missing chunks

17.1.2 Leaking keys

17.2 Frequently made programming mistakes

17.2.1 Failure to block (aka wait)

17.2.2 Calling an MRTask2 from inside another MRTask2

17.2.3 Running out of Fork/Join threads

18 Things H2O does not Support

It’s worth pointing out the following list of items that are available in some other popular languages and frameworks. Some of these are in the H2O roadmap and some are not a fit for H2O.

18.1 Row names

Unlike R Data Frames, H2O does not support naming individual rows. Allowing this is one factor that inhibits the R runtime from scaling well. Don’t expect H2O to ever support this.

18.2 Unique per-row strings for many rows

H2O currently turns columns with small numbers of unique strings into an Enum, and turns columns with large numbers of unique strings into N/As.

In the future, H2O will be able to read in columns with large numbers of strings and treat a separate “String” datatype as a top-level datatype. These will be unusable for modeling purposes, but flow through to be able to print the value as an output. For some applications, this may serve as an adequate substitute for row names.

18.3 High availability (HA)

H2O is currently vulnerable to single-node failures rendering the entire cluster inoperable. Since H2O has an In-Memory architecture, the response to this is to manually kill and restart the cluster and any jobs that were in progress. Proper HA support is on the roadmap, but a (perhaps sufficient) step on the way there is the ability to checkpoint and restart model building.

19 Glossary

Categorical
Cluster (of H2O nodes)
Domain
DKV
DRemoteTask
Enum
Fluid Vector (FV)
Freezable
Future
Fork/Join
H2OCountedCompleter
Iced
In-Core
In-Memory
Job
Key
Level
MRTask2
Node (in a Cluster)
Out-of-Core
REST API endpoint
UKV
ValueArray (VA or sometimes just Array)
Vector Group