H2O and R

Jessica Lanford

11/14/14

1 What is H2O?

It is the only alternative to combine the power of highly advanced algorithms, the freedom of open source, and the capacity of truly scalable in-memory processing for big data on one of many nodes. Combined, these capabilities make it faster, easier, and more cost-effective to harness big data to maximum benefit for the business.

Data collection is easy. Decision making is hard. H2O makes it fast and easy to derive insights from your data through faster and better predictive modeling. Existing Big Data stacks are batch-oriented. Search and analytics need to be interactive. Use machines to learn machine-generated data. And more data beats better algorithms.

With H2O, you can make better predictions by harnessing sophisticated, ready-to-use algorithms and the processing power you need to analyze bigger data sets, more models, and more variables.

Get started with minimal effort and investment. H2O is an extensible open source platform that offers the most pragmatic way to put big data to work for your business. With H2O, you can work with your existing languages and tools. Further, you can extend the platform seamlessly into your Hadoop environments. Get H2O!

Download H2O http://www.h2o.ai/download

Join the Community h2ostream@googlegroups.com and github.com/0xdata/h2o.git

2 Introduction

This documentation describes the functionality of R in H2O. Further information on H2O's system and algorithms, as well as R user documentation, can be found at the H2O website at http://docs.h2o.ai. This introductory section describes how H2O works with R, followed by a brief overview of generalized linear models (GLM).

R requires a reference object to the H2O instance because it uses a REST API to send functions to H2O. Data sets are not transmitted directly through the REST API. Instead, the user sends a command (containing an HDFS path to the data set, for example) either through the browser or via the REST API to ingest data from disk.

The data set is then assigned a Key in H2O that the user may refer to in future commands to the web server. After preparing your dataset for modeling by defining the significant data and removing the insignificant data, you can create models to represent the results of the data analysis. One of the most popular models for data analysis is GLM.

GLM estimates regression models for outcomes following exponential distributions in general. In addition to the Gaussian (i.e. normal) distribution, these include Poisson, binomial, gamma and Tweedie distributions. Each serves a different purpose, and depending on distribution and link function choice, it can be used either for prediction or classification.

H2O supports Spark, YARN, and all versions of Hadoop. Hadoop is a scalable open-source file system that uses clusters to enable distributed storage and processing of datasets. Depending on your data size, you can get started on your desktop or scale using multiple nodes with Hadoop.

H2O nodes run as JVM invocations on Hadoop nodes. (Note that, for performance reasons, we recommend you avoid running an H2O node on the same hardware as the Hadoop NameNode if it can be avoided.)

Since H2O nodes run as mapper tasks in Hadoop, administrators can see them in the normal JobTracker and TaskTracker frameworks. This provides process-level (i.e. JVM instance-level) visibility.

H2O helps R users make the leap from laptop-based processing to large-scale environments. Hadoop helps H2O users scale their data processing capabilities based on their current needs. Using H2O, R, and Hadoop, you can create a complete end-to-end data analysis solution. For more information about H2O on Hadoop, refer to http://docs.h2o.ai/bits/hadoop/H2O_on_Hadoop_0xdata.pdf.

This document will walk you through the four steps to data analysis with H2O: installing H2O, preparing your data for modeling (data munging), creating a model using state-of-the-art machine learning algorithms, and scoring your models.

3 Installation

To use H2O with R, you can start H2O outside of R and connect to it, or you can launch H2O from R. However, if you launch H2O from R and close the R session, the H2O instance is closed as well. The client object is used to direct R to datasets and models located in H2O.

3.1 Installing R or R Studio

To download R:

1. Go to http://cran.r-project.org/mirrors.html.
2. Select your closest local mirror.
3. Select your operating system (Linux, OS X, or Windows).
4. Depending on your OS, download the appropriate file, along with any required packages.

5. When the download is complete, unzip the file and install.

To download R Studio:

1. Go to http://www.rstudio.com/products/rstudio/.
2. Select your deployment type (desktop or server).
3. Download the file.
4. When the download is complete, unzip the file and install.

3.2 Installing H2O in R

1. Load the latest CRAN H2O package by running

   ```
   install.packages("h2o")
   ```

   Note: Our push to CRAN will be behind the bleeding edge version and due to resource constraints, may be behind the published version. However, there is a best-effort to keep the versions the same.

   To get the latest build, download it from http://h2o.ai/download and make sure to run the following (replacing the asterisks [*] with the version number):

   ```
   >if ("package:h2o" %in% search()) { detach("package:h2o", unload=TRUE) }
   >if ("h2o" %in% rownames(installed.packages())) { remove.packages("h2o") }
   >install.packages("h2o", repos=(c("http://s3.amazonaws.com/h2o-release/h2o/master/****/
   R", getOption("repos"))))
   >library(h2o)
   ```

2. If you are operating on a single node, initialize H2O using

   ```
   h2o_server = h2o.init()
   ```

   To connect with an existing H2O cluster node other than the default localhost:54321, specify the IP address and port number in the parentheses. For example:

   ```
   h2o_cluster = h2o.init(ip = "192.555.1.123", port = 12345)
   ```

3. Run a demo to see an example classification model using GLM.

   ```
   demo(glm)
   ```

3.3 Making a build from Source Code

1. If you are a developer who wants to make changes to the R package before building and installing it, pull the source code from Git (https://github.com/0xdata/h2o) and follow the instructions in From Source Code (Github) at http://docs.h2o.ai/developuser/quickstart_git.html.

After making the build, navigate to the Rcran folder with the R package in the builds directory, then run and install.

```
$ R CMD INSTALL h2o_2.7.0.99999.tar.gz
* installing to library 'C:/Users//Documents/R/win-library/3.0'
* installing *source* package 'h2o' ...
** R
** demo
** inst
...
*** installing help indices
** building package indices
** testing if installed package can be loaded
...
DONE (h2o)
```

2. Verify that H2O installed properly:

```
library(h2o)
localH2O = h2o.init()
```

4 Data Preparation in R

The following section describes some important points to remember about data preparation (munging) and some of the tools and methods available in H2O, as well as a data training example.

4.1 Notes

- Although it may seem like you are manipulating the data in R due to the look and feel, once the data has been passed to H2O, all data munging occurs in the H2O instance. The information is passed to R through JSON APIs.
- You are not limited by R's ability to handle data, but by the total amount of memory allocated to the H2O instance. To process large data sets, make sure to allocate enough memory. For more information, refer to "Launching in R."
- Be aware that its possible to manipulate datasets with thousands of factor levels using H2O in R, so if you ask H2O to display a table in R with information from high cardinality factors, the results may overwhelm R's capacity.
- To manipulate data in R and not in H2O, use `as.data.frame()`, `as.h2o()`, and `str()`.
  - `as.data.frame( )` converts the current data into an R data frame. Be aware that if your request exceeds Rs capabilities due to the amount of data, the R session will

crash. If possible, we recommend only taking the necessary data columns, and not the whole data set.

- `as.h2o( )` transfers data from R to the H2O instance. We recommend ensuring that you allocate enough memory to the H2O instance for successful data transfer.
- `str( )` confirms that the data transferred correctly. Its a good way to verify there were no data loss or conversion issues.

4.2 Tools and Methods

The following section describes some of the tools and methods available in H2O for data preparation.

- **Data Profiling**: Quickly summarize the shape of your dataset to avoid bias or missing information before you start building your model. Missing data, zero values, text, and a visual distribution of the data are visualized automatically upon data ingestion.
- **Summary Statistics**: Visualize your data with summary statistics to get the mean, standard deviation, min, max, cardinality, quantile, and a preview of the data set.
- **Aggregate, Filter, Bin, and Derive Columns**: Build unique views with Group functions, Filtering, Binning, and Derived Columns.
- **Slice, Log Transform, and Anonymize**: Normalize, anonymize, and partition to get your data into the right shape for modeling.
- **Variable Creation**: Highly customizable variable value creation to hone in on the key data characteristics to model.
- **PCA**: Principal Component Analysis makes feature selection easy with a simple to use interface and standard input values.
- **Training and Validation Sampling Plan**: Design a random or stratified sampling plan to generate data sets for model training and scoring.

4.3 Demo: Splitting Data for Training

The following section depicts an example of data training using `ddply()`. Using this method, you can split your dataset and apply a function to the subsets.

To apply a user-specified function to each subset of an H2O dataset and combine the results, use `ddply()`, with the name of the H2O object, the variable name, and the function in the parentheses. For more information about functions, refer to `h2o.addFunction` in the Appendix.

```
library(h2o)
localH2O = h2o.init()

# Import iris dataset to H2O
irisPath = system.file("extdata", "iris_wheader.csv", package = "h2o")
```

```
iris.hex = h2o.importFile(localH2O, path = irisPath, key = "iris.hex")

# Add function taking mean of sepal_len column
fun = function(df) { sum(df[,1], na.rm = T)/nrow(df) }
h2o.addFunction(localH2O, fun)

# Apply function to groups by class of flower
# uses h2o's ddply, since iris.hex is an H2OParsedData object
res = ddply(iris.hex, "class", fun)
head(res)
```

5 Models

The following section describes the features and functions of some common models available in H2O. For more information about running these models in R using H2O, refer to "Running Models."

H2O supports the following models: Deep Learning, Generalized Linear Models (GLM), Gradient Boosted Regression (GBM), K-Means, Naïve Bayes, Principal Components Analysis (PCA), Principal Components Regression (PCR), Random Forest (RF), and Cox Proportional Hazards (PH).

The list is growing quickly, so check back often at www.h2o.ai to see the latest additions. The following list describes some common model types and features.

Generalized Linear Models (GLM): A flexible generalization of ordinary linear regression for response variables that have error distribution models other than a normal distribution. GLM unifies various other statistical models, including linear, logistic, Poisson, and more.

Decision trees: Used in RF; a decision support tool that uses a tree-like graph or model of decisions and their possible consequences.

Gradient Boosting (GBM): A method to produce a prediction model in the form of an ensemble of weak prediction models. It builds the model in a stage-wise fashion and is generalized by allowing an arbitrary differentiable loss function. It is one of the most powerful methods available today.

K-Means: A method to uncover groups or clusters of data points often used for segmentation. It clusters observations into k certain points with the nearest mean.

Anomaly Detection: Identify the outliers in your data by invoking a powerful pattern recognition model.

Deep Learning: Model high-level abstractions in data by using non-linear transformations in a layer-by-layer method. Deep learning is an example of supervised learning and can make use of unlabeled data that other algorithms cannot.

Naïve Bayes: A probabilistic classifier that assumes the value of a particular feature is unrelated to the presence or absence of any other feature, given the class variable. It is often used in text categorization.

Grid Search: The standard way of performing hyper-parameter optimization to make model configuration easier. It is measured by cross-validation of an independent data set.

After creating a model, use it to make predictions. For more information about predictions, refer to "Predictions."

5.1 Demo: GLM

The following demo demonstrates how to import a file, define significant data, view data, create testing and training sets using sampling, define the model, and display the results.

```
## Import dataset and display summary
> airlinesURL = "https://s3.amazonaws.com/h2o-airlines-unpacked/allyears2k.csv"
> airlines.hex = h2o.importFile(localH2O, path = airlinesURL, key = "airlines.hex")
> summary(airlines.hex)

## Define columns to ignore, quantiles and histograms
> high_na_counts = h2o.ignoreColumns(data = airlines.hex)
> high_na_counts
[1] "AirTime"           "TaxiIn"            "TaxiOut"           "CancellationCode"
[5] "CarrierDelay"      "WeatherDelay"      "NASDelay"          "SecurityDelay"
[9] "LateAircraftDelay"

## Find number of flights by airport
> originFlights = h2o.ddply(airlines.hex, 'Origin', nrow)
> originFlights.R = as.data.frame(originFlights)

## Find number of cancellations per month
> flightByMonth = h2o.ddply(airlines.hex,"Month", nrow)
> end = Sys.time()
> time_to_aggre_h2o = end - start

## Find months with the highest cancellation ratio
> fun = function(df) {sum(df$Cancelled)}
> h2o.addFunction(h, fun)
> cancellationsByMonth = h2o.ddply(airlines.hex,"Month", fun)
> cancellation_rate = cbind(flightByMonth$Month,cancellationsByMonth$C1
/flightByMonth$C1)

# Construct test and train sets using sampling
> airline.split = h2o.splitFrame(data = airlines.hex,ratios = 0.85)
> airline.train = airline.split[[1]]
> airline.test = airline.split[[2]]
```

```
# Display a summary using table-like functions
> summary(as.factor(airline.train$Cancelled))
> summary(as.factor(airline.test$Cancelled))

# Set predictor and response variables
> Y = "IsDepDelayed"
> X = c("Origin", "Dest", "DayofMonth", "Year", "UniqueCarrier", "DayOfWeek", "Month",
"DepTime", "ArrTime", "Distance")
# Define the data for the model and display the results
> airlines.glm<- h2o.glm(data=airlines.hex, x=X, y=Y, family = "binomial", nfolds
= 1)
> airlines.glm
```

6 Data Manipulation in R

The following section describes some common R commands. For a complete command list, including parameters, refer to http://docs.h2o.ai/bits/h2o_package.pdf. For additional help within R's Help tab, precede the command with a question mark (for example, `?h2o`) for suggested commands containing the search terms. For more information on a command, precede the command with two question marks (`??h2o`).

6.1 Launching in R

If you do not specify the argument `max_mem_size` when you run `h2o.init( )`, the default heap size of the H2O instance running on 32-bit Java is 1g. H2O checks the Java version and suggests an upgrade if you are running 32-bit Java. On 64-bit Java, the heap size is 1/4 of the total memory available on the machine.

For best performance, the allocated memory should be 4x the size of your data, but never more than the total amount of memory on your computer. For larger data sets, running on a server or service with more memory available for computing is recommended.

To launch H2O from R, run the following in R:

```
> library(h2o) ##Loads required files for H2O
localH2O <- h2o.init(ip = 'localhost', port = 54321, nthreads= -1, max_mem_size =
4g') ##Starts H2O on the localhost, port 54321, with 4g of memory using all CPUs
on the host
```

R displays the following output:

```
Successfully connected to http://localhost:54321
       R is connected to H2O cluster:
   H2O cluster uptime:         11 minutes 35 seconds
   H2O cluster version:        2.7.0.1497
   H2O cluster name:           H2O_started_from_R
```

```
  H2O cluster total nodes:    1
  H2O cluster total memory:   3.56 GB
  H2O cluster total cores:    8
  H2O cluster allowed cores:  8
  H2O cluster healthy:        TRUE
```

6.2 Launching from the Command Line

After launching the H2O instance, initialize the connection by running `h2o.init( )` with the IP address and port number of a node in the cluster. In the following example, change 192.168.1.161to your local host.

```
> library(h2o)
> localH2O <- h2o.init(ip = '192.168.1.161', port =54321)
```

6.3 Checking Cluster Status

To check the status and health of the H2O cluster, use `h2o.clusterInfo( )`.

```
> library(h2o)
> localH2O = h2o.init(ip = 'localhost', port = 54321)
> h2o.clusterInfo(localH2O)
```

An easy-to-read summary of information about the cluster displays.

```
R is connected to H2O cluster:
  H2O cluster uptime:         43 minutes 43 seconds
  H2O cluster version:        2.7.0.1497
  H2O cluster name:           H2O_started_from_R
  H2O cluster total nodes:    1
  H2O cluster total memory:   3.56 GB
  H2O cluster total cores:    8
  H2O cluster allowed cores:  8
  H2O cluster healthy:        TRUE
```

6.4 Importing Files

The H2O package consolidates all of the various supported import functions using `h2o.importFile( )`. Although `h2o.importFolder` and `h2o.importHDFS` will still work, these functions are deprecated and should be updated to `h2o.importFile( )`.

```
## To import small iris data file from H2O's package:
> irisPath = system.file("extdata", "iris.csv", package="h2o")
> iris.hex = h2o.importFile(localH2O, path = irisPath, key = "iris.hex")
      |=============================================| 100%
```

```
## To import an entire folder of files as one data object:
> pathToFolder = "/Users/Amy/data/airlines/"
> airlines.hex = h2o.importFile(localH2O, path = pathToFolder, key = "airlines.hex")
      |=================================================| 100%

## To import from HDFS, connect to your Hadoop cluster and start an H2O instance
in R using the IP that was specified by Hadoop:
> remoteH2O = h2o.init(ip= <IPAddress>, port =54321)
> pathToData = "hdfs://mr-0xd6.0xdata.loc/datasets/airlines_all.csv"
> airlines.hex = h2o.importFile(remoteH2O, path = pathToData, key = "airlines.hex")
      |=================================================| 100%
```

6.5 Uploading Files

To upload a file from your local disk, we recommend `h2o.importFile`. However, uploadFile will still work. In the parentheses, specify the H2O reference object in R and the complete URL or normalized file path for the file.

```
> irisPath = system.file("extdata", "iris.csv", package="h2o")
> iris.hex = h2o.uploadFile(localH2O, path = irisPath, key = "iris.hex")
|=================================================| 100%
```

6.6 Finding Factors

To determine if any column in a data set is a factor, use h2o.anyFactor( ) with the name of the R reference object in the parentheses.

```
> irisPath = system.file("extdata", "iris_wheader.csv", package="h2o")
> iris.hex = h2o.importFile(localH2O, path = irisPath)
|=================================================| 100%
> h2o.anyFactor(iris.hex)
[1] TRUE
```

6.7 Converting Data Frames

To convert an H2O parsed data object into an R data frame that can be manipulated using R commands, use `as.data.frame( )` with the name of the R reference object in the parentheses.

Caution: While this can be very useful, be careful using this command when converting H2O parsed data objects. H2O can easily handle data sets that are often too large to be handled equivalently well in R.

```
> prosPath <- system.file("extdata", "prostate.csv", package=h2o")
##Creates object that defines path
```

```
 > prostate.hex = h2o.importFile(localH2O, path = prosPath)
##Imports data set
  |=================================================| 100%

 > prostate.data.frame<- as.data.frame(prostate.hex)
##Converts current data frame (prostate data set) to an R data frame
 > summary(prostate.data.frame) ##Displays summary of data frame
       ID             CAPSULE            AGE             RACE
Min.   :  1.00   Min.   :0.0000   Min.   :43.00   Min.   :0.000
1st Qu.: 95.75   1st Qu.:0.0000   1st Qu.:62.00   1st Qu.:1.000
       ....
```

6.8 Converting to Factors

To convert an integer into a non-ordered factor (also called an enum or categorical), use `as.factor( )` with the name of the R reference object in parentheses followed by the number of the column to convert in brackets.

```
> prosPath = system.file("extdata", "prostate.csv", package="h2o")
##Creates object that defines path
> prostate.hex = h2o.importFile(localH2O, path = prosPath)
##Imports data set
|=================================================| 100%
> prostate.hex[,4] = as.factor(prostate.hex[,4])
##Converts column 4 (RACE) to an enum
> summary(prostate.hex)
 ID               CAPSULE          AGE             RACE    DPROS
 Min.   :  1.00   Min.   :0.0000   Min.   :43.00   1 :341  Min.   :1.000
```

6.9 Transferring Data Frames

To transfer a data frame from the R environment to the H2O instance, use `as.h2o( )`. In the parentheses, specify the name of the h2o.init object that communicates with R and H2O and the object in the R environment to be converted to an H2O object. Optionally, you can include the reference to the H2O instance (the key). Precede the key with `key=` and enclose the key in quotes as in the following example.

```
> as.h2o(localH2O, df, key= "dataframe.h2o")
##Converts R object "df" to H2O object dataframe.h2o"
  |=========================================================| 100%
IP Address: localhost
Port      : 54321
Parsed Data Key: dataframe.h2o

  X1 X2 X3 X4 X5 X6 X7 X8 X9 X10 index
1  1  1  1  1  1  1  1  1  1   1     1
```

```
2  2  2  2  2  2  2  2  2  2   2     1
3  3  3  3  3  3  3  3  3  3   3     1
4  4  4  4  4  4  4  4  4  4   4     1
5  5  5  5  5  5  5  5  5  5   5     1
6  6  6  6  6  6  6  6  6  6   6     1
```

6.10 Creating Hex Keys

To create a hex key on the server running H2O for data sets that were manipulated in R, use `h2o.assign( )`. For instance, in the following example, the prostate data set was uploaded to the H2O instance and the data was manipulated to remove outliers. `h2o.assign( )` saves the new data set on the H2O server so that it can be analyzed using H2O without overwriting the original data set.

```
> prosPath = system.file("extdata", "prostate.csv", package=h2o")
##Creates object that defines path
> prostate.hex = h2o.importFile(localH2O, path = prosPath)
##Imports data set
    |=================================================| 100%
> prostate.qs = quantile(prostate.hex$PSA)
> PSA.outliers = prostate.hex[prostate.hex$PSA
<= prostate.qs[2] | prostate.hex$PSA >=   prostate.qs[10],]
> PSA.outliers = h2o.assign(PSA.outliers, "PSA.outliers")
> nrow(prostate.hex)
[1] 380
> nrow(PSA.outliers)
[1] 380
```

6.11 Getting Column Names

To obtain a list of the column names in the data set, use `colnames( )` with the name of the R reference object in the parentheses.

```
> colnames(iris.hex)
[1] "C1" "C2" "C3" "C4" C5" ##Displays the titles of the columns
```

6.12 Getting Minimum and Maximum Values

To obtain the maximum values for the real-valued columns in a data set, use `max( )` with the name of the R reference object in the parentheses. To obtain the minimum values for the real-valued columns in a data set, use `min( )` with the name of the R reference object in the parentheses.

```
> min(prostate.hex)
[1] 0
```

```
> max(prostate.hex)
[1] 380
```

6.13 Getting Quantiles

To request quantiles for an H2O parsed data set, use `quantile( )` with the name of the R reference object in the parentheses. To request a quantile for a single numerical column, use `quantile(ReferenceObject$ColumnName)`, where `ReferenceObject` represents the R reference object name and `ColumnName` represents the name of the specified column. When you request for a full parsed data set consisting of a single column, `quantile( )` displays a matrix with quantile information for the data set.

```
> quantile(breadth.hex)
  0%  25%  50%  75% 100%
   1    2    3    6   33
> quantile(breadth.hex$C1)
  0%  25%  50%  75% 100%
   1    2    3    6   33
```

6.14 Summarizing Data

To generate a summary (similar to the one in R) for each of the columns in the data set, use `summary( )` with the name of the R reference object in the parentheses. For continuous real functions, this produces a summary that includes information on quartiles, min, max, and mean. For factors, this produces information about counts of elements within each factor level.

```
> summary(australia.hex)
 premax          salmax         minairtemp      maxairtemp      maxsst
 Min.   : 18.0   Min.   :3441   Min.   :272.6   Min.   :285.0   Min.   :285697
 1st Qu.: 75.0   1st Qu.:3490   1st Qu.:277.0   1st Qu.:292.0   1st Qu.:290491
 Median :150.0   Median :3533   Median :278.8   Median :299.9   Median :293643
 Mean   :161.5   Mean   :3529   Mean   :279.9   Mean   :297.5   Mean   :295676
 3rd Qu.:250.0   3rd Qu.:3558   3rd Qu.:282.0   3rd Qu.:302.4   3rd Qu.:301942
 Max.   :450.0   Max.   :3650   Max.   :290.0   Max.   :310.0   Max.   :303697
 maxsoilmoist     Max_czcs         runoffnew
 Min.   : 0.000   Min.   : 0.160   Min.   :   0.0
 1st Qu.: 0.000   1st Qu.: 0.629   1st Qu.:   0.0
 Median : 4.000   Median : 1.020   Median :  19.0
 Mean   : 5.117   Mean   : 1.369   Mean   : 232.2
 3rd Qu.: 9.000   3rd Qu.: 1.705   3rd Qu.: 300.0
 Max.   :16.000   Max.   :11.370   Max.   :2400.0
```

6.15 Summarizing Data in a Table

To summarize the data, use `h2o.table( )`. Because H2O can handle larger data sets, it is possible to generate tables that are larger than R's capacity. To minimize this risk, `h2o.table` is called inside of a `head( )` or `tail( )` command. Within `head( )` and `tail( )`, specify the column numbers to summarize. To summarize multiple columns, use `head(h2o.table (ObjectName[, c(ColumnNumber,ColumnNumber)]))` where `ObjectName` is the name of the object in R and `ColumnNumber` is the number of the column.

```
library(h2o)
localH2O = h2o.init()
prosPath = system.file("extdata", "prostate.csv", package="h2o")
prostate.hex = h2o.importFile(localH2O, path = prosPath, key = "prostate.hex")
summary(prostate.hex)

# Counts of the ages of all patients
head(h2o.table(prostate.hex[,3]))
h2o.table(prostate.hex[,3], return.in.R = TRUE)

# Two-way table of ages (rows) and race (cols) of all patients
head(h2o.table(prostate.hex[,c(3,4)]))
h2o.table(prostate.hex[,c(3,4)], return.in.R = TRUE)
```

6.16 Generating Random Uniformly Distributed Numbers

To append a column of random numbers to an H2O data frame for facilitating creation of testing/training data splits for analysis and validation in H2O, use `h2o.runif()` with the name of the R reference object in the parentheses. This method is best for customized frame splitting; otherwise, use `h2o.splitFrame()`. However, `h2o.runif()` is not as fast or stable as `h2o.splitFrame()`.

```
> s = h2o.runif(phbirths.hex) ##Creates object s" for h2o.runif on phbirths data
set
> summary (s)  ##Summarize the results of h2o.runif
 rnd
 Min.   :0.000105
 1st Qu.:0.249472
 Median :0.489969
 Mean   :0.492103
 3rd Qu.:0.739377
 Max.   :0.998859
> phbirths.train = phbirths.hex[s <= 0.8,] ##Create training set with threshold of
0.8
> phbirths.train = h2o.assign(phbirths.train, phbirths.train") ##Assign name to training
set
```

```
> phbirths.test = phbirths.hex[s > 0.8,] ##Create test set with threshold to filter
values greater than 0.8
> phbirths.test = h2o.assign(phbirths.test, phbirths.test") ##Assign name to test
set
> nrow(phbirths.train) + nrow(phbirths.test) ##Combine results of test & training
sets, then display result
[1] 1115
```

6.17 Splitting Frames

To generate two subsets (according to specified ratios) from an existing H2O data set for testing/training, use `h2o.splitFrame()`. This method is preferred over `h2o.runif` because it is faster and more stable.

```
prostate.split = h2o.splitFrame(data = prostate.hex , ratios = 0.75)
##Splits data in prostate data frame with a ratio of 0.75
prostate.train = prostate.split[1]
##Creates training set from 1st data set in split
prostate.test = prostate.split[2]
##Creates training set from 1st data set in split
```

6.18 Getting Frames

To create a reference object to the data frame in H2O, use `h2o.getFrame()`. This is helpful for users that alternate between the web UI and the R API or multiple users accessing the same H2O instance. The following example assumes prostate.hex is in the key-value (KV) store.

```
> prostate.hex = h2o.getFrame(h2o = localH2O, key = "prostate.hex")
```

6.19 Getting Models

To create a reference object for the model in H2O, use `h2o.getModel()`. This is helpful for users that alternate between the web UI and the R API or multiple users accessing the same H2O instance. The following example assumes GLMModel is in the key-value (KV) store.

```
glm.model = h2o.getModel(h2o = localH2O, key = "GLMModel")
```

6.20 Listing H2O Objects

To generate a list of all H2O objects generated during a session, along with each objects size in bytes, use `h2o.ls()` with the address of the instance in the parentheses. If the instance is local, use localH2O.

```
> h2o.ls(localH2O)
                                                    Key Bytesize
```

```
1               GBM_8e4591a9b413407b983d73fbd9eb44cf    40617
2               GBM_a3ae2edf5dfadbd9ba5dc2e9560c405d     1516
```

6.21 Removing H2O Objects

To remove an H2O object on the server associated with an object in the R environment, use `h2o.rm()`. For optimal performance, we recommend removing the object from the R environment as well using `remove()`, with the name of the object in the parentheses. If you do not specify an R environment, then the current environment is used.

```
> h2o.rm(object= localH2O, keys= "prostate.train")
```

6.22 Adding Functions

To add a user-defined function in R to the H2O instance, use `h2o.addFunction()`, with the IP address of the H2O instance and the function in the parentheses.

```
library(h2o)
localH2O = h2o.init()
h2o.addFunction(localH2O, function(x) { 2*x + 5 }, "simpleFun")
```

7 Running Models

To run the models, use the following commands.

7.1 Gradient Boosted Models (GBM)

To generate gradient boosted models for developing forward-learning ensembles, use `h2o.gbm()`. In the parentheses, define x (the predictor variable vector), y (the integer or categorical response variable), the distribution type (multinomial is the default, gaussian is used for regression), and the name of the H2OParsedData object.

```
> h2o.gbm(y = dependent, x = independent, data = australia.hex,
> n.trees = 10, interaction.depth = 3,
   n.minobsinnode = 2, shrinkage = 0.2, distribution= "gaussian")
     |================================================| 100%
 Mean-squared Error by tree:
  [1] 230760.11 166957.80 124904.30  94031.17  72367.01  57180.17  47092.85
  [8]  39168.05  34456.00  31095.86  28397.10
```

To generate a classification model that uses labels, use `distribution= "multinomial"`:

```
> h2o.gbm(y = dependent, x = independent, data = australia.hex, n.trees
= 15, interaction.depth = 5,
```

```
 n.minobsinnode = 2, shrinkage = 0.01, distribution= "multinomial")

Confusion matrix:
Reported on australia1.hex
Predicted
Actual     0 3 6 7 14 16 17 19 20 25 38 43 61 75 82 107 138 150 167 191 200
  0      115 0 0 0  0  0  0  0  0  0  0  0  0  0  0   0   0   0   0   0   0
. . .
```

7.2 Generalized Linear Models (GLM)

To generate a generalized linear model for developing linear models for exponential distributions, use `h2o.glm()`. You can apply regularization to the model by adjusting the lambda and alpha parameters.

```
> prostate.hex = h2o.importFile(localH2O, path =
"https://raw.github.com/0xdata/h2o/master/smalldata/logreg/prostate.csv",
key = "prostate.hex")
       |=====================================================| 100%
> h2o.glm(y = "CAPSULE", x = c("AGE","RACE","PSA","DCAPS"), data =
prostate.hex, family = "binomial", nfolds = 10, alpha = 0.5)
 |=========================================================| 100%
    Coefficients:
   AGE      RACE     DCAPS       PSA Intercept
      -0.01104  -0.63136   1.31888   0.04713  -1.10896

     Normalized Coefficients:
   AGE      RACE     DCAPS       PSA Intercept
      -0.07208  -0.19495   0.40972   0.94253  -0.33707

     Degrees of Freedom: 379 Total (i.e. Null);  375 Residual
     Null Deviance:     514.9
     Residual Deviance: 461.3  AIC: 471.3
     Deviance Explained: 0.10404
     AUC: 0.68875  Best Threshold: 0.328
    Confusion Matrix:
     Predicted
     Actual   false true Error
     false    127  100 0.441
     true      51  102 0.333
     Totals   178  202 0.397

     Cross-Validation Models:
      Nonzeros       AUC Deviance Explained
     Model 1         4 0.6532738        0.048419803
     Model 2         4 0.6316527       -0.006414532
```

```
      Model 3         4 0.7100840        0.087779178
      Model 4         4 0.8268698        0.243020554
      Model 5         4 0.6354167        0.153190735
      Model 6         4 0.6888889        0.041892118
      Model 7         4 0.7366071        0.164717509
      Model 8         4 0.6711310        0.004897310
      Model 9         4 0.7803571        0.200384622
      Model 10        4 0.7435897        0.114548543
```

7.3 K-Means

To generate a K-Means model for data characterization, use `h2o.kmeans()`. This algorithm does not rely on a dependent variable.

```
> prostate.km = h2o.kmeans(data = prostate.hex, centers = 10,
cols = c("AGE", "RACE", "VOL", "GLEASON"))
  |=======================================================| 100%
print(prostate.km)
IP Address: 127.0.0.1
Port      : 54321
Parsed Data Key: prostate6.hex
K-Means Model Key: KMeans2_99fea55be4a22f741df74532d7844bb4
K-means clustering with 10 clusters of sizes 41, 27, 59, 17, 21, 47, 26, 61, 47,
34

      Cluster means:
      AGE     RACE         VOL  GLEASON
      1  69.73171 1.024390 37.99756098 6.512195
      2  54.48148 1.111111  0.32222222 6.518519
      3  62.59322 1.067797  0.19322034 5.966102
      .....
```

7.4 Principal Components Analysis (PCA)

To map a set of variables onto a subspace using linear transformations, use `h2o.prcomp()`. This is the first step in Principal Components Regression.

```
> ausPath = system.file("extdata", "australia.csv", package="h2o")
> australia.hex = h2o.importFile(localH2O, path = ausPath)
  |=======================================================| 100%
> australia.pca = h2o.prcomp(data = australia.hex, standardize = TRUE)
  |=======================================================| 100%

 Standard deviations:
 1.750703 1.512142 1.031181 0.8283127 0.6083786 0.5481364 0.4181621 0.2314953
      ....
```

```
summary(australia.pca)
Importance of components:
....
```

7.5 Principal Components Regression (PCR)

To map a set of variables to a set of linearly independent variables, use `h2o.pcr()`. The variables in the new set are linearly independent linear combinations of the original variables and exist in a lower-dimension subspace. This transformation is prepended to the regression model to improve results.

```
prostate.hex = h2o.importURL(localH2O, path = paste("https://raw.github.com",
  "0xdata/h2o/master/smalldata/logreg/prostate.csv", sep = "/"), key = "prostate.hex")
h2o.pcr(x = c("AGE","RACE","PSA","DCAPS"), y = "CAPSULE", data = prostate.hex, family
= "binomial",
  nfolds = 0, alpha = 0.5, ncomp = 2)
```

7.6 Predictions

The following section describes some of the prediction methods available in H2O.

Predict: Generate outcomes of a data set with any model. Predict with GLM, GBM, Decision Trees or Deep Learning models.

Confusion Matrix: Visualize the performance of an algorithm in a table to understand how a model performs.

Area Under Curve (AUC): A graphical plot to visualize the performance of a model by its sensitivity, true positive, or false positive to select the best model.

Hit Ratio: A classification matrix to visualize the ratio of the number of correctly classified and incorrectly classified cases.

PCA Score: Determine how well your feature selection is for a particular model.

Multi-Model Scoring: Compare and contrast multiple models on a data set to find the best performer to deploy into production.

To apply an H2O model to a holdout set for predictions based on model results, use `h2o.predict()`. In the following example, H2O generates a model and then displays the predictions for that model.

```
> prostate.fit = h2o.predict(object = prostate.glm, newdata = prostate.hex)
> (prostate.fit)

predict        X0        X1
      1       0 0.7452267 0.2547732
      2       1 0.3969807 0.6030193
      3       1 0.4120950 0.5879050
      4       1 0.3726134 0.6273866
      5       1 0.6465137 0.3534863
      6       1 0.4331880 0.5668120
```

8 Support

There are multiple ways to request support for H2O:

Email: support@h2o.ai

H2OStream on Google Groups: https://groups.google.com/d/forum/h2ostream

JIRA: http://jira.0xdata.com/

Meetup information: http://h2o.ai/events

9 References

R Package: http://docs.h2o.ai/bits/h2o_package.pdf

R Ensemble documentation: http://www.stat.berkeley.edu/ ledell/R/h2oEnsemble.pdf

Slide deck: http://h2o.ai/blog/2013/08/big-data-science-in-h2o-with-r/

R project website: http://www.r-project.org

10 Appendix: Commands

The following section lists some common commands by function that are available in R and a brief description of each command.

10.1 Data Set Operations

Data Import/Export

`h2o.downloadCSV`: Download a H2O dataset to a CSV file on local disk.

`h2o.exportFile`: Export H2O Data Frame to a File.

`h2o.importFile`: Import a file from the local path and parse it.

`h2o.parseRaw`: Parse a raw data file.

`h2o.uploadFile`: Upload a file from the local drive and parse it.

Native R to H2O Coercion

`as.h2o`: Convert an R object to an H2O object

H2O to Native R Coercion

`as.data.frame`: Check if an object is a data frame, or coerce it if possible.

`as.matrix`: Convert the specified argument to a matrix.

`as.table`: Build a contingency table of the counts at each combination of factor levels.

Data Generation

`h2o.createFrame`: Create an H2O data frame, with optional randomization.

`h2o.runif`: Produce a vector of random uniform numbers.

`h2o.interaction`: Create interaction terms between categorical features of an H2O Frame.

Data Sampling / Splitting

`h2o.sample`: Sample an existing H2O Frame by number of observations.

`h2o.splitFrame`: Split an existing H2O data set according to user-specified ratios.

`h2o.nFoldExtractor`: Split an existing H2O data set into N folds and return a specified holdout split, and the rest.

Missing Data Handling

`h2o.impute`: Impute a column of data using the mean, median, or mode.

`h2o.insertMissingValue`: Replaces a user-specified fraction of entries in a H2O dataset with missing values.

`h2o.ignoreColumns`: Returns columns' names of a parsed H2O data object that are recommended to be ignored in an analysis per the specified ratio in `max_na`.

10.2 General Data Operations

Subscripting example to pull pieces from data object.

```
x[i]
x[i, j, ... , drop = TRUE]
x[[i]]
x$name

x[i] <- value
x[i, j, ...] <- value
x[[i]] <- value
x$i <- value
```

Subsetting

`head, tail`: Return the First or Last Part of an Object

Concatenation

`c`: Combine Values into a Vector or List

`cbind`: Take a sequence of H2O datasets and combine them by column.

Data Attributes

`colnames`: Return column names for a parsed H2O data object.

`colnames<-`: Retrieve or set the row or column names of a matrix-like object.

`names`: Get the name of an object.

`names<-`: Set the name of an object.

`dim`: Retrieve the dimension of an object.

`length`: Get the length of vectors (including lists) and factors.

`nrow`: Return a count of the number of rows in an H2OParsedData object.

`ncol`: Return a count of the number of columns in an H2OParsedData object.

`h2o.anyFactor`: Check if an H2O parsed data object has any categorical data columns.

`is.factor`: Check if a given column contains categorical data.

Data Type Coercion

`as.factor`: Convert a column from numeric to factor.

`as.Date`: Converts a column from factor to date.

10.3 Methods from Group Generics

Math (H2O)

`abs`: Compute the absolute value of x.

`sign`: Return a vector with the signs of the corresponding elements of x (the sign of a real number is 1, 0, or -1 if the number is positive, zero, or negative, respectively).

`sqrt`: Computes the principal square root of x, $\sqrt{x}$.

`ceiling`: Take a single numeric argument x and return a numeric vector containing the smallest integers not less than the corresponding elements of x.

`floor`: Take a single numeric argument x and return a numeric vector containing the largest integers not greater than the corresponding elements of x.

`trunc`: Take a single numeric argument x and return a numeric vector containing the integers formed by truncating the values in x toward 0.

`log`: Compute logarithms (by default, natural logarithms).

`exp`: Compute the exponential function.

Math (generic)

`cummax`: Display a vector whose elements are the cumulative maxima of the elements of the argument.

`cummin`: Display a vector whose elements are the cumulative minima of the elements of the argument.

`cumprod`: Display a vector whose elements are the cumulative products of the elements of the argument.

`cumsum`: Display a vector whose elements are the cumulative sums of the elements of the argument.

`log10`: Compute common (i.e., base 10) logarithms

`log2`: Compute binary (i.e., base 2) logarithms.

`log1p`: Compute log(1+x) accurately also for $|x| << 1$.

`acos`: Compute the trigonometric arc-cosine.

`acosh`: Compute the hyperbolic arc-cosine.

`asin`: Compute the trigonometric arc-sine.

`asinh`: Compute the hyperbolic arc-sine.

`atan`: Compute the trigonometric arc-tangent.

`atanh`: Compute the hyperbolic arc-tangent.

`expm1`: Compute exp(x) - 1 accurately also for $|x| << 1$.

`cos`: Compute the trigonometric cosine.

`cosh`: Compute the hyperbolic cosine.

`cospi`: Compute the trigonometric two-argument arc-cosine.

`sin`: Compute the trigonometric sine.

`sinh`: Compute the hyperbolic sine.

`sinpi`: Compute the trigonometric two-argument arc-sine.

`tan`: Compute the trigonometric tangent.

`tanh`: Compute the hyperbolic tangent.

`tanpi`: Compute the trigonometric two-argument arc-tangent.

`gamma`: Display the gamma function $\gamma(x)$

`lgamma`: Display the natural logarithm of the absolute value of the gamma function.

`digamma`: Display the first derivative of the logarithm of the gamma function.

`trigamma`: Display the second derivative of the logarithm of the gamma function.

Math2 (H2O)

`round`: Round the values in its first argument to the specified number of decimal places (default 0).

`signif`: Round the values in its first argument to the specified number of significant digits.

Summary (H2O)

`max`: Display the maximum of all the input arguments.

`min`: Display the minimum of all the input arguments.

`range`: Display a vector containing the minimum and maximum of all the given arguments.

`sum`: Calculate the sum of all the values present in its arguments.

Summary (generic)

`prod`: Display the product of all values present in its arguments.

`any`: Given a set of logical vectors, determine if at least one of the values is true.

`all`: Given a set of logical vectors, determine if all of the values are true.

10.4 Other Aggregations

Non-Group Generic Summaries `mean`: Generic function for the (trimmed) arithmetic mean.

`sd`: Calculate the standard deviation of a column of continuous real valued data.

`var`: Compute the variance of x.

`summary`: Produce result summaries of the results of various model fitting functions.

`quantile`: Obtain and display quantiles for H2O parsed data.

Row / Column Aggregation `apply`: Apply a function over an H2O parsed data object (an array).

Group By Aggregation `h2o.ddply`: Split H2O dataset, apply a function (defined in `h2o.addFunction`), and display results.

`h2o.addFunction`: Add a function defined in R to the H2O server for future use.

Tabulation

`h2o.table`: Use the cross-classifying factors to build a table of counts at each combination of factor levels.

10.5 Data Munging

Transformation Framework

`h2o.exec`: Directly transmit and execute an R expression in the H2O console.

General Column Manipulations

`is.na`: Display missing elements.

`unique`: Display a vector, data frame, or array with duplicate elements/rows removed.

Element Index Selection

`findInterval`: Find Interval Numbers or Indices.

`which`: Display the row numbers for which the condition is true.

Conditional Element Value Selection

`ifelse`: Apply conditional statements to numeric vectors in H2O parsed data objects.

Numeric Column Manipulations

`h2o.cut`: Convert H2O Numeric Data to Factor.

`diff`: Display suitably lagged and iterated differences.

Character Column Manipulations

`strsplit`: Splits the given factor column on the input split.

`tolower`: Change the elements of a character vector to lower case.

`toupper`: Change the elements of a character vector to lower case.

`trim`: Remove leading and trailing white space.

`h2o.gsub`: Match a pattern and replaces all instances of the matched pattern with the replacement string globally.

`h2o.sub`: Match a pattern and replace the first instance of the matched pattern with the replacement string.

Factor Level Manipulations

`levels`: Display a list of the unique values found in a column of categorical data.

`revalue`: Replace specified values with new values in a factor or character vector.

Date Manipulations

`h2o.month`: Convert the entries of a H2OParsedData object from milliseconds to months (on a 0 to 11 scale).

`h2o.year`: Convert the entries of a H2OParsedData object from milliseconds to years, indexed starting from 1900.

Matrix Operations

`%*%`: Multiply two matrices, if they are conformable.

`t`: Given a matrix or data.frame x, t returns the transpose of x.

10.6 Data Modeling

Model Training

`h2o.coxph`: Fit a Cox Proportional Hazards Model.

`h2o.gbm`: Build gradient boosted classification trees and gradient boosted regression trees on a parsed data set.

`h2o.glm`: Fit a generalized linear model, specified by a response variable, a set of predictors, and a description of the error distribution.

`h2o.kmeans`: Perform k-means clustering on a data set.

`h2o.naiveBayes`: Build gradient boosted classification trees and gradient boosted regression trees on a parsed data set.

`h2o.pcr`: Run GLM regression on PCA results, and allow for transformation of test data to match PCA transformations of training data.

`h2o.prcomp`: Perform principal components analysis on the given data set.

`h2o.randomForest`: Perform random forest classification on a data set.

Deep Learning

`h2o.deeplearning`: Perform Deep Learning neural networks on an H2OParsedData object.

`h2o.anomaly`: Detect anomalies in a H2O dataset using a H2O deep learning model with auto-encoding.

`h2o.deepfeatures`: Extract the non-linear features from a H2O dataset using a H2O deep learning model.

Model Scoring

`h2o.predict`: Obtain predictions from various fitted H2O model objects.

Classification Model Helpers

`h2o.confusionMatrix`: Display prediction errors for classification data from a column of predicted responses and a column of actual (reference) responses in H2O.

`h2o.gains`: Construct the gains table and lift charts for binary outcome algorithms.

`h2o.hitRatio`: Compute the percentage of instances where the actual class of an observation is in the top user-specified number of classes predicted by the model.

`h2o.performance`: Evaluate the predictive performance of a model via various measures.

Clustering Helper

`h2o.gapStatistic`: Measure the suitability of the fit of a clustering algorithm.

Regression Model Helper

`h2o.mse`: Display the mean squared error calculated from a column of predicted responses and a column of actual (reference) responses in H2O.

GLM Helper

`h2o.getGLMLambdaModel`: Retrieve the H2O GLM model built using a specific value of lambda from a lambda search.

10.7 H2O Cluster Operations

H2O Key Value Store Access

`h2o.assign`: Assign H2O hex.keys to objects in their R environment.

`h2o.getFrame`: Get a reference to an existing H2O data set.

`h2o.getModel`: Get a reference to an existing H2O model.

`h2o.ls: Display` a list of object keys in the running instance of H2O.

`h2o.rm`: Remove H2O objects from the server where the instance of H2O is running, but does not remove it from the R environment.

H2O Object Serialization

`h2o.loadAll`: Load all H2OModel object in a directory from disk that was saved using `h2o.saveModel` or `h2o.saveAll`.

`h2o.loadModel`: Load an H2OModel object from disk.

`h2o.saveAll`: Save all H2OModel objects to disk to be loaded back into H2O using `h2o.loadModel` or `h2o.loadAll`.

`h2o.saveModel`: Save an H2OModel object to disk to be loaded back into H2O using `h2o.loadModel`.

H2O Cluster Connection
`h2o.init`: Connect to a running H2O instance and check the local H2O R package is the correct version.

`h2o.shutdown`: Shut down the specified H2O instance. All data on the server will be lost!

H2O Load Balancing

`h2o.rebalance`: Rebalance (repartition) an existing H2O data set into given number of chunks

(per Vec), for load-balancing across multiple threads or nodes.

H2O Cluster Information

`h2o.clusterInfo`: Display the name, version, uptime, total nodes, total memory, total cores and health of a cluster running H2O.

`h2o.clusterStatus`: Retrieve information on the status of the cluster running H2O.

H2O Logging
`h2o.clearLogs`: Clear all H2O R command and error response logs from the local disk.

`h2o.downloadAllLogs`: Download all H2O log files to the local disk.

`h2o.logAndEcho`: Write a message to the H2O Java log file and echo it back.

`h2o.openLog`: Open existing logs of H2O R POST commands and error responses on the local disk.

`h2o.getLogPath`: Get the file path for the H2O R command and error response logs.

`h2o.setLogPath`: Set the file path for the H2O R command and error response logs.

`h2o.startLogging`: Begin logging H2O R POST commands and error responses.

`h2o.stopLogging`: Stop logging H2O R POST commands and error responses.