Gradient Boosted Machines with H2O's R Package

November 2014

Contents

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 or 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/h2oai/h2o.git`

2 Introduction

This vignette presents the gradient boosted machine (GBM) framework in the H2O package at `http://cran.r-project.org/web/packages/h2o/index.html`. Further documentation on H2O's system and algorithms can be found at the h2o.ai website at `http://docs.h2o.ai` and the R package manual at `http://cran.r-project.org/web/packages/h2o/h2o.pdf`. The full datasets and code of this vignette can be found at the H2O Github at `github.com/h2oai/h2o.git`. This introductory section provides instructions on getting H2O started, followed by a brief overview of gradient boosting.

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)
   ```

   To see `h2o.gbm` at work, run the following command to observe an automatic demo of an example classification model built using H2O's GBM.

   ```
   demo(h2o.gbm)
   ```

3.3 Making a build from Source Code

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/h2oai/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.

```
$ make clean
$ make build
$ cd target/R/src/contrib
$ R CMD INSTALL h2o_2.9.0.99999.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)
```

4 H2O Initialization

4.1 Launching from 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, we recommend running on a server or service with more memory available for computing.

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
```

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)
```

4.2 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 192.168.1.161 to your local host.

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

4.3 Launching on Hadoop

To launch H2O nodes and form a cluster on the Hadoop cluster, run:

```
$ hadoop jar h2odriver_hdp2.1.jar water.hadoop.h2odriver -libjars ../h2o.jar -mapperXmx
1g -nodes 1 -output hdfsOutputDirName
```

- For each major release of each distribution of hadoop, there is a driver jar file that the user will need to launch H2O with. Currently available driver jar files in each build of H2O includes `h2odriver_cdh5.jar, h2odriver_hdp2.1.jar`, and `mapr2.1.3.jar`.
- The above command launches exactly one 1g node of H2O; however, we recommend launching the cluster with 4 times the memory of your data file.
- `mapperXmx` is the mapper size or the amount of memory allocated to each node.
- `nodes` is the number of nodes requested to form the cluster.
- `output` is the name of the directory created each time a H2O cloud is created so it is necessary for the name to be unique each time it is launched.

4.4 Launching on an EC2

Launch the EC2 instances using the H2O AMI by running `h2o-cluster-launch-instances.py`.

```
$ python h2o-cluster-launch-instances.py
Using boto version 2.27.0
Launching 2 instances.
Waiting for instance 1 of 2 ...
  .
  .
  instance 1 of 2 is up.
Waiting for instance 2 of 2 ...
  instance 2 of 2 is up.
```

4.5 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
```

4.6 Support

Users of the H2O package may submit general enquiries and bug reports to h2o.ai support at h2ostream@googlegroups.com. Alternatively, specific bugs or issues may be filed to the h2o.ai JIRA at https://0xdata.atlassian.net/secure/Dashboard.jspa.

4.7 Gradient Boosting overview

A gradient boosted model can be either regression or classification. Both are forward-learning ensemble methods that obtain predictive results through gradually improved estimations. Boosting is a flexible nonlinear regression procedure that helps improve the accuracy of trees. By sequentially applying weak classification algorithms to the incrementally changed data, a series of decision trees are created that produce an ensemble of weak prediction models.

While boosting trees increases their accuracy, it also decreases speed and interpretability. The gradient boosting method generalizes tree boosting to minimize these issues.

After creating a GBM, H2O displays the confusion matrix that shows the classifications for each group, the associated error by group, and the overall average error.

4.7.1 Summary of features

H2O's GBM functionalities include:

- supervised learning for regression and classification tasks
- distributed and parallelized computation on either a single node or a multi-node cluster
- fast and memory-efficient Java implementations of the underlying algorithms
- elegant web interface to mirror the model building and scoring process running in R
- grid search for hyperparameter optimization and model selection
- model export in plain java code for deployment in production environments
- additional parameters for model tuning

4.7.2 Theory and framework

Gradient boosting is a machine learning technique that combines two powerful tools: gradient-based optimization and boosting. Gradient-based optimization uses gradient computations in order to minimize a model's loss function with respect to the training data. Boosting additively collects an ensemble of weak models in order to ultimately create a strong learning system for predictive tasks. Here we consider gradient boosting in the example of K-class classification, although the model for regression follows similar logic. The following analysis follows from the discussion in Hastie et al (2010) at http://statweb.stanford.edu/~tibs/ElemStatLearn/.

GBM for classification

```
1. Initialize $f_{k0} = 0, k = 1, 2, \ldots, K$
2. For $m = 1$ to $M$
    a. Set $p_k(x) = \frac{e^{f_k(x)}}{\sum_{l=1}^{K} e^{f_l(x)}}$ for all $k = 1, 2 \ldots, K$
    b. For $k = 1$ to $K$
        i. Compute $r_{ikm} = y_{ik} - p_k(x_i), i = 1, 2, \ldots, N$
        ii. Fit a regression tree to the targets $r_{ikm}, i = 1, 2, \ldots, N$,
            giving terminal regions $R_{jim}, 1, 2, \ldots, J_m$
        iii. Compute
             $$\gamma_{jkm} = \frac{K-1}{K} \frac{\sum_{x_i \in R_{jkm}} (r_{ikm}}{\sum_{x_i \in R_{jkm}} |r_{ikm}|(1 - |r_{ikm}|)}, j = 1, 2, \ldots, J_m$$
        iv. Update $f_{km}(x) = f_{k,m-1}(x) + \sum_{j=1}^{J_m} \gamma_{jkm} I(x \in R_{jkm})$
3. Output $\hat{f_k}(x) = f_{kM}(x), k = 1, 2, \ldots, K$
```

In the above algorithm, the index m tracks of the number of weak learners added to the current ensemble. Within this outer loop, there is an inner loop across each of the K classes. In this inner loop, the first step is to compute the residuals, r_{ikm}, which are actually the gradient values, for each of the N bins in the CART model, and then to fit a regression tree to these gradient computations. This fitting process is distributed and parallelized, and details on this framework can be found on the h2o.ai blog at http://h2o.ai/blog/2013/10/building-distributed-gbm-h2o/.

The final procedure in the inner loop is to add to the current model to the fitted regression tree, which improves the accuracy of the model during the inherent gradient descent step. After M iterations, the final "boosted" model can be tested out on new data.

4.7.3 Key parameters

In the above example, an important user-specified value is N, which represents the number of bins that data are partitioned into before the tree's best split point is determined. Split points are determined by considering the end points of each bin, and the one-versus-many split for each bin. To model all factors individually, you can specify high N values, but this will slow down the modeling process. For shallow trees, we recommend keeping the total count of bins across all splits at 1024 (so that a top-level split uses 1024, but a 2nd level split uses 512 bins, and so forth). This value is then maxed with the input bin count.

Another important parameter to specify is the size J of the trees, which must be controlled in order to avoid overfitting. Increasing J enables larger variable interaction effects, so knowing about these effects is helpful in setting the value for J. Large values of J have also been found to have excessive computational cost, since Cost = #columns $\cdot N \cdot K \cdot 2^J$. However, lower values generally also have the highest performance. Models with $4 \leq J \leq 8$ and a larger number of trees M reflect this generalization. Later, we will discuss how to use grid search models to tune these parameters in the model selection process.

You can also specify the shrinkage constant, which controls the learning rate of the model and is actually a form of regularization. Shrinkage modifies the algorithm's update of $f_{km}(x)$ instead with the scaled addition $\nu \cdot \sum_{j=1}^{J_m} \gamma_{jkm} I(x \in R_{jkm})$, where the constant ν is between 0 and 1. Smaller values of ν lead to greater rates of training errors, assuming that M is fixed, but that in general ν and M are inversely related when the error is fixed. However, despite the greater rate of training error with small values of ν, very small values ($\nu < 0.1$) typically lead to better generalization and performance on test data.

5 Use case: Classification with Airline data

5.1 Airline dataset overview

Download the Airline dataset from: https://github.com/h2oai/h2o/blob/master/smalldata/airlines/allyears2k_headers.zip and save the .csv file to your working directory. Before running the Airline demo, review how to load data with H2O.

5.1.1 Loading data

Loading a dataset in R for use with H2O is slightly different from the usual methodology, as we must convert our datasets into `H2OParsedData` objects. For this example, download the toy weather dataset from https://raw.githubusercontent.com/h2oai/h2o/master/smalldata/weather.csv. Load the data to your current working directory in your R Console (do this for any future dataset downloads), and then run the following command.

```
weather.hex = h2o.uploadFile(h2o_server, path = "weather.csv", header = TRUE, sep
= ",", key = "weather.hex")
```

To see a brief summary of the data, run the following command.

```
summary(weather.hex)
```

5.2 Performing a trial run

Returning to the Airline dataset, load the dataset with H2O and select the variables to use to predict a chosen response. For example, model whether flights are delayed based on the departure's scheduled day of the week and day of the month.

```
#Load the data and prepare for modeling
air_train.hex = h2o.uploadFile(h2o_server, path = "AirlinesTrain.csv", header = TRUE,
sep = ",", key = "airline_train.hex")

air_test.hex = h2o.uploadFile(h2o_server, path = "AirlinesTest.csv", header = TRUE,
sep = ",", key = "airline_test.hex")

myX <- c("fDayofMonth", "fDayOfWeek")
```

Now, train the GBM model:

```
air.model <- h2o.gbm(y = "IsDepDelayed", x = myX,
                   distribution="multinomial",
                   data = air_train.hex, n.trees=100,
                   interaction.depth=4,
                   shrinkage=0.1,
                   importance=TRUE)
```

Since it is meant just as a trial run, the model contains only 100 trees. In this trial run, no validation set was specified, so by default, the model evaluates the entire training set. To use n-fold validation, specify, for example, `nfolds=5`.

5.2.1 Extracting and handling the results

Now, extract the parameters of the model, examine the scoring process, and make predictions on new data.

```
#View the specified parameters of your GBM model
air.model@model$params

#Examine the performance of the trained model
air.model
```

The second command (`air.model`) returns the trained model's training and validation errors.

After generating a satisfactory model, use the `h2o.predict()` command to compute and store predictions on new data, which can then be used for further tasks in the interactive modeling process.

```
#Perform classification on the held out data
prediction = h2o.predict(air.model, newdata=air_test.hex)

#Copy predictions from H2O to R
pred = as.data.frame(prediction)

head(pred)
```

5.3 Web interface

To mirror the model building process in R, use the H2O web interface. After loading data or training a model in R, use the browser to access the IP address and port number (e.g., `localhost:12345`) and launch the web interface. From the web interface, click Admin > Jobs to view specific model details or click Data > View All to view and keep track of currently used datasets.

5.3.1 Variable importances

To enable the variable importances option, use the additional argument `importance=TRUE`. This displays the absolute and relative predictive strength of each feature in the prediction task. From R, access these strengths using the command `air.model@model$varimp`. You can also view a visualization of the variable importances on the web interface.

5.3.2 Java model

To access Java (POJO) code to use to build the current model in Java, click Java model button in the top right of a model summary page. If the model is small enough, the code for the model displays within the GUI; larger models can be inspected after downloading the model.

To download the model:

1. Open the terminal window.
2. Create a directory where the model will be saved.
3. Set the new directory as the working directory.
4. Follow the curl and java compile commands displayed in the instructions at the top of the Java model.

5.4 Grid search for model comparison

To support grid search capabilities for model tuning, specify sets of values for parameter arguments to tweak certain parameters and observe changes in model behavior. The following is an example of a grid search:

```
air.grid <- h2o.gbm(y = "IsDepDelayed", x = myX,
                    distribution="multinomial",
                    data = air_train.hex, n.trees=c(5,10,15),
                    interaction.depth=c(2,3,4),
                    shrinkage=c(0.1,0.2))
```

This example specifies three different tree numbers, three different tree sizes, and two different shrinkage values. This grid search model effectively trains eighteen different models over the possible combinations of these parameters. Of course, sets of other parameters can be specified for a larger space of models. This allows for more subtle insights in the model tuning and selection process, especially during inspection and comparison of the trained models after the grid search process is complete. To decide how and when to choose different parameter configurations in a grid search, refer to the beginning section for parameter descriptions and suggested values.

```
#print out all prediction errors and run times of the models
air.grid
air.grid@model

#print out a *short* summary of each of the models (indexed by parameter)
air.grid@sumtable

#print out *full* summary of each of the models
all_params = lapply(air.grid@model, function(x) { x@model$params })
all_params

#access a particular parameter across all models
shrinkages = lapply(air.grid@model, function(x) { x@model$params$shrinkage })

shrinkages
```

6 Conclusion

Gradient boosted machines sequentially fit new models to provide a more accurate estimate of a response variable in supervised learning tasks such as regression and classification. Though notorious for being difficult to distribute and parallelize, H2O's GBM offers both features in its framework, along with a straightforward environment for model tuning and selection.

7 References

Dietterich, Thomas G, and Eun Bae Kong. **Machine Learning Bias, Statistical Bias, and Statistical Variance of Decision Tree Algorithms.** http://www.iiia.csic.es/~vtorra/tr-bias.pdf ML-95 255 (1995).

Elith, Jane, John R Leathwick, and Trevor Hastie. **A Working Guide to Boosted Regression Trees.** http://onlinelibrary.wiley.com/doi/10.1111/j.1365-2656.2008.01390.x/abstract Journal of Animal Ecology 77.4 (2008): 802-813

Friedman, Jerome H. **Greedy Function Approximation: A Gradient Boosting Machine.** http://statweb.stanford.edu/~jhf/ftp/trebst.pdf Annals of Statistics (2001): 1189-1232.

Friedman, Jerome, Trevor Hastie, Saharon Rosset, Robert Tibshirani, and Ji Zhu. **Discussion of Boosting Papers.**http://web.stanford.edu/~hastie/Papers/boost_discussion.pdf Ann. Statist 32 (2004): 102-107

Friedman, Jerome, Trevor Hastie, and Robert Tibshirani. **"Additive Logistic Regression: A Statistical View of Boosting (With Discussion and a Rejoinder by the Authors).** http://projecteuclid.org/DPubS?service=UI&version=1.0&verb=Display&handle=euclid.aos/1016218223 The Annals of Statistics 28.2 (2000): 337-407

Hastie, Trevor, Robert Tibshirani, and J Jerome H Friedman. **The Elements of Statistical Learning** http://statweb.stanford.edu/~tibs/ElemStatLearn/printings/ESLII_print10.pdf. Vol.1. N.p., page 339: Springer New York, 2001.

Niu, Feng, et al. **Hogwild!: A lock-free approach to parallelizing stochastic gradient descent.** https://papers.nips.cc/paper/4390-hogwild-a-lock-free-approach-to-parallelizing-stochastic-gradient-descent.pdf Advances in Neural Information Processing Systems 24 (2011): 693-701. (algorithm implemented is on p.5)