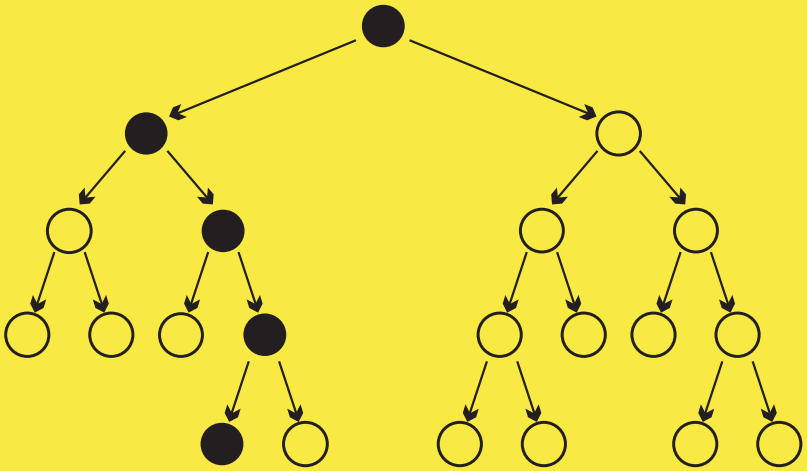
H2O.ai

GRADIENT BOOSTED MODELS WITH H2O'S R PACKAGE

Cliff Click, Jessica Lanford, Michal Malohlava, Viraj Parmar & Hank Roark



Published by H2O © 2015

Gradient Boosted Models with H2O's R Package

Cliff Click Jessica Lanford Michal Malohlava
Viraj Parmar Hank Roark

---

http://h2o.ai/resources/

September 2015: Third Edition

Gradient Boosted Models with H2O's R Package
by Cliff Click, Jessica Lanford,
Michal Malohlava, Viraj Parmar,
& Hank Roark

Published by H2O.ai, Inc.
2307 Leghorn St.
Mountain View, CA 94043

September 2015: Third Edition

Printed in the United States of America.

Contents

1 Introduction

This document describes how to use Gradient Boosted Models (GBM) with H2O. Examples are written in R and Python. Topics include:

- installation of H2O
- basic GBM concepts
- building GBM models in H2O
- interpreting model output
- making predictions

2 What is H2O?

H2O is fast, scalable, open-source machine learning and deep learning for smarter applications. With H2O, enterprises like PayPal, Nielsen Catalina, Cisco, and others can use all their data without sampling to get accurate predictions faster. Advanced algorithms such as deep learning, boosting, and bagging ensembles are built-in to help application designers create smarter applications through elegant APIs. Some of our initial customers have built powerful domain-specific predictive engines for recommendations, customer churn, propensity to buy, dynamic pricing, and fraud detection for the insurance, healthcare, telecommunications, ad tech, retail, and payment systems industries.

Using in-memory compression, H2O handles billions of data rows in-memory, even with a small cluster. To make it easier for non-engineers to create complete analytic workflows, H2O's platform includes interfaces for R, Python, Scala, Java, JSON, and CoffeeScript/JavaScript, as well as a built-in web interface, Flow. H2O is designed to run in standalone mode, on Hadoop, or within a Spark Cluster, and typically deploys within minutes.

H2O includes many common machine learning algorithms, such as generalized linear modeling (linear regression, logistic regression, etc.), Naïve Bayes, principal components analysis, k-means clustering, and others. H2O also implements best-in-class algorithms at scale, such as distributed random forest, gradient boosting and deep learning. Customers can build thousands of models and compare the results to get the best predictions.

H2O is nurturing a grassroots movement of physicists, mathematicians, and computer scientists to herald the new wave of discovery with data science by collaborating closely with academic researchers and industrial data scientists. Stanford university giants Stephen Boyd, Trevor Hastie, Rob Tibshirani advise the H2O team on building scalable machine learning algorithms. With hundreds

of meetups over the past three years, H2O has become a word-of-mouth phenomenon, growing amongst the data community by a hundred-fold, and is now used by 30,000+ users and is deployed using R, Python, Hadoop, and Spark in 2000+ corporations.

Try it out

- Download H2O directly at http://h2o.ai/download.
- Install H2O's R package from CRAN at https://cran.r-project.org/web/packages/h2o/.
- Install the Python package from PyPI at https://pypi.python.org/pypi/h2o/.

Join the community

- To learn about our meetups, training sessions, hackathons, and product updates, visit http://h2o.ai.
- Visit the open source community forum at https://groups.google.com/d/forum/h2ostream.
- Join the chat at https://gitter.im/h2oai/h2o-3.

3 Installation

The easiest way to directly install H2O is via an R or Python package.

(**Note**: The examples in this document were created with H2O version 3.2.0.1.)

3.1 Installation in R

To load a recent H2O package from CRAN, run:

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

Note: The version of H2O in CRAN is often one release behind the current version.

For the latest recommended version, download the latest stable H2O-3 build from the H2O download page:

1. Go to http://h2o.ai/download.
2. Choose the latest stable H2O-3 build.
3. Click the "Install in R" tab.

4. Copy and paste the commands into your R session.

After H2O is installed on your system, verify the installation:

```
library(h2o)

#Start H2O on your local machine using all available
    cores.
#By default, CRAN policies limit use to only 2 cores.
h2o.init(nthreads = -1)

#Get help
?h2o.glm
?h2o.gbm

#Show a demo
demo(h2o.glm)
demo(h2o.gbm)
```

3.2 Installation in Python

To load a recent H2O package from PyPI, run:

```
pip install h2o
```

To download the latest stable H2O-3 build from the H2O download page:

1. Go to http://h2o.ai/download.
2. Choose the latest stable H2O-3 build.
3. Click the "Install in Python" tab.
4. Copy and paste the commands into your Python session.

After H2O is installed, verify the installation:

```
import h2o

# Start H2O on your local machine
h2o.init()

# Get help
help(h2o.glm)
```

```
help(h2o.gbm)

# Show a demo
h2o.demo("glm")
h2o.demo("gbm")
```

3.3 Pointing to a Different H2O Cluster

The instructions in the previous sections create a one-node H2O cluster on your local machine.

To connect to an established H2O cluster (in a multi-node Hadoop environment, for example) specify the IP address and port number for the established cluster using the `ip` and `port` parameters in the `h2o.init()` command. The syntax for this function is identical for R and Python:

```
h2o.init(ip = "123.45.67.89", port = 54321)
```

3.4 Example Code

R and Python code for the examples in this document are available here: https://github.com/h2oai/h2o-3/blob/master/h2o-docs/src/booklets/v2_2015/source/gbm

The document source itself can be found here: https://github.com/h2oai/h2o-3/blob/master/h2o-docs/src/booklets/v2_2015/source/GBM_Vignette.tex

3.5 Citation

To cite this booklet, use the following:

Click, C., Lanford, J., Malohlava, M., Parmar, V., and Roark, H. (Aug. 2015). *Gradient Boosted Models with H2O*. http://h2o.ai/resources/.

4 Overview

A GBM is an ensemble of either regression or classification tree models. 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. Weak classification algorithms are sequentially applied to the incrementally changed data to create a series of decision trees, producing an ensemble of weak prediction models.

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

4.1 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
- the ability to run H2O from R, Python, Scala, or the intuitive web UI (Flow)
- grid search for hyperparameter optimization and model selection
- model export in plain Java code for deployment in production environments
- additional parameters for model tuning (for a complete listing of parameters, refer to the Model Parameters section.)

Gradient boosted models (also known as gradient boosting machines) sequentially fit new models to provide a more accurate estimate of a response variable in supervised learning tasks such as regression and classification. Although GBM is known to be difficult to distribute and parallelize, H2O provides an easily distributable and parallelizable version of GBM in its framework, as well as an effortless environment for model tuning and selection.

4.2 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 to minimize a model's loss function in terms of the training data.

Boosting additively collects an ensemble of weak models to create a robust learning system for predictive tasks. The following example considers gradient boosting in the example of K-class classification; the model for regression follows a 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 $R_{jkm}, 1, 2, \ldots, J_m$
      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 for multi-class classification, H2O builds k-regression trees: one tree represents each target class. 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.

Within 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. A regression tree is then fit to these gradient computations. This fitting process is distributed and parallelized. Details on this framework are available at http://h2o.ai/blog/2013/10/building-distributed-gbm-h2o/.

The final procedure in the inner loop is to add the current model to the fitted regression tree to improve 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.

4.3 Distributed Trees

H2O's implementation of GBM uses distributed trees. H2O overlays trees on the data by assigning a tree node to each row. The nodes are numbered and the number of each node is stored as `Node_ID` in a temporary vector for each row. H2O makes a pass over all the rows using the most efficient method, which may not necessarily be numerical order.

A local histogram using only local data is created in parallel for each row on each node. The histograms are then assembled and a split column is selected to make the decision. The rows are re-assigned to nodes and the entire process is repeated.

For example, with an initial tree, all rows start on node 0. An in-memory MapReduce (MR) task computes the statistics and uses them to make an algorithmically-based decision, such as lowest mean squared error (MSE). In the next layer in the tree (and the next MR task), a decision is made for each row: if X $<$ 1.5, go right in the tree; otherwise, go left. H2O computes the stats for each new leaf in the tree, and each pass across all the rows builds the entire layer.

For multinomial or binomial, the split is determined by the number of bins. The number of bins is evaluated to find the best split out of the possible combinations. For example, for a hundred-column dataset that uses twenty bins, there are 2000 (20x100) possible split points.

Each layer represents another MR task: a tree that is five layers deep requires five passes. Each tree level is fully data-parallelized. Each pass builds a per-node histogram in the MR call over one layer in the tree. During each pass, H2O analyzes the tree level and decides how to build the next level. In another pass, H2O reassigns rows to new levels by merging the two passes and then builds a histogram for each node. Each per-level histogram is done in parallel.

Scoring and building is done in the same pass. Each row is tested against the decision from the previous pass and assigned to a new leaf, where a histogram is built. To score, H2O traverses the tree and obtains the results. The tree is compressed to a smaller object that can still be traversed, scored, and printed.

Although the GBM algorithm builds each tree one level at a time, H2O is able to quickly run the entire level in parallel and distributed. The processing requirements for more data can be offset by more CPUs or nodes. Since H2O

does the per-level compute in parallel, which requires sending histograms over the network, the amount of data can become very large for a very deep tree.

For the MSE reports, the zero-tree report uses the class distribution as the prediction. The one-tree report uses the first tree, so the first two reports are not equal. The reported MSE is the inclusive effect of all prior trees and generally decreases monotonically on the training dataset. However, the curve will generally bottom out and then begin to slowly rise on the validation set as overfitting sets in.

The computing cost is based on the number of leaves, but depending on the dataset, the number of leaves can be difficult to predict. The maximum number of leaves is 2^d, where d represents the tree depth.

4.4 Treatment of Factors

If the training data contains columns with categorical levels (factors), then these factors are split by assigning an integer to each distinct categorical level, then binning the ordered integers according to the user-specified number of bins `nbins_cats` (which defaults to 1024 bins), and then picking the optimal split point among the bins.

To specify a model that considers all factors individually (and perform an optimal group split, where every level goes in the right direction based on the training response), specify `nbins_cats` to be at least as large as the number of factors. Values greater than 1024 (the maximum number of levels supported in R) are supported, but might increase model training time.

The value of `nbins_cats` for categorical factors has a much greater impact on the generalization error rate than `nbins` for real- or integer-valued columns (where higher values mainly lead to more accurate numerical split points). For columns with many factors, a small `nbins_cats` value can add randomness to the split decisions (since the columns are grouped together somewhat arbitrarily), while large values can lead to perfect splits, resulting in overfitting.

4.5 Key Parameters

In the above example, an important user-specified value is N, which represents the number of bins used to partition the data before the tree's best split point is determined. To model all factors individually, 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 for numerical columns (so that a top-level split uses 1024, but a second-level split uses 512 bins, and so forth). This value is then maxed with the input bin count.

Specify the size of the trees (J) to avoid overfitting. Increasing J results in larger variable interaction effects. Large values of J have also been found to have an excessive computational cost, since Cost $= \#\text{columns} \cdot N \cdot K \cdot 2^J$. Lower values generally have the highest performance.

Models with $4 \leq J \leq 8$ and a larger number of trees M reflect this generalization. Grid search models can be used to tune these parameters in the model selection process. For more information, refer to Grid Search for Model Comparison.

To control the learning rate of the model, specify the `learn_rate` constant, which is actually a form of regularization. Shrinkage modifies the algorithm's update of $f_{km}(x)$ 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 constant. In general, ν and M are inversely related when the error rate is constant. However, despite the greater rate of training error with small values of ν, very small ($\nu < 0.1$) values typically lead to better generalization and performance on test data.

5 Use Case: Airline Data Classification

Download the Airline dataset from: https://github.com/h2oai/h2o-2/blob/master/smalldata/airlines/allyears2k_headers.zip and save the .csv file to your working directory.

5.1 Loading Data

Loading a dataset in R or Python for use with H2O is slightly different from the usual methodology because the datasets must be converted into `H2OParsedData` objects. For this example, download the toy weather dataset from https://github.com/h2oai/h2o-2/blob/master/smalldata/weather.csv.

Example in R
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.

```
1 library(h2o)
2 h2o.init()
3 weather.hex <- h2o.uploadFile(path  = "weather.csv",
    header  = TRUE, sep = ",", destination_frame = "
    weather.hex")
4
```

```
# To see a brief summary of the data, run the
    following command.
summary(weather.hex)
```

Example in Python
Load the data to your current working directory in Python (do this for any future dataset downloads), and then run the following command.

```
import h2o
h2o.init()
weather_hex = h2o.import_file("weather.csv")

# To see a brief summary of the data, run the
    following command.
weather_hex.describe()
```

5.2 Performing a Trial Run

Load the Airline dataset into H2O and select the variables to use to predict the response. The following example models delayed flights based on the departure's scheduled day of the week and day of the month.

Example in R

```
# Load the data and prepare for modeling
airlines.hex <- h2o.uploadFile(path = "allyears2k_
    headers.csv", header = TRUE, sep = ",",
    destination_frame = "airlines.hex")

# Generate random numbers and create training,
    validation, testing splits
r <- h2o.runif(airlines.hex)
air_train.hex <- airlines.hex[r  < 0.6,]
air_valid.hex <- airlines.hex[(r >= 0.6) & (r < 0.9),]
air_test.hex  <- airlines.hex[r  >= 0.9,]

myX <- c("DayofMonth", "DayOfWeek")

# Now, train the GBM model:
air.model <- h2o.gbm(y = "IsDepDelayed", x = myX,
    distribution="bernoulli", training_frame = air_
```

```
    train.hex, validation_frame = air_valid.hex,
    ntrees=100, max_depth=4, learn_rate=0.1)
```

Example in Python

```
# Load the data and prepare for modeling
airlines_hex = h2o.import_file(path = "
    allyears2k_headers.csv")

# Generate random numbers and create training,
    validation, testing splits
r = airlines_hex.runif()    # Random UNIForm numbers,
    one per row
air_train_hex = airlines_hex[r  < 0.6]
air_valid_hex = airlines_hex[(r >= 0.6) & (r < 0.9)]
air_test_hex  = airlines_hex[r  >= 0.9]

myX = ["DayofMonth", "DayOfWeek"]

# Now, train the GBM model:
air_model = h2o.gbm(y = "IsDepDelayed", x = myX,
    distribution="bernoulli", training_frame =
    air_train_hex, validation_frame = air_valid_hex,
    ntrees=100, max_depth=4, learn_rate=0.1)
```

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 an n-folds value (for example, `nfolds=5`).

5.3 Extracting and Handling the Results

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

Example in R

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

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

Example in Python

```
# View the specified parameters of your GBM model
air_model.params

# Examine the performance of the trained model
air_model
```

The first 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 the new data, which can then be used for further tasks in the interactive modeling process.

Example in R

```
# 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)
```

Example in Python

```
# Perform classification on the held out data
prediction = air_model.predict(air_test_hex)

# Copy predictions from H2O to Python
pred = prediction.as_data_frame()

pred.head()
```

5.4 Web Interface

H2O users have the option of using an intuitive web interface for H2O, Flow. After loading data or training a model, point your browser to your IP address and port number (e.g., localhost:12345) to launch the web interface. In the web UI, click Admin > Jobs to view specific details about your model or click Data > List All Frames to view all current H2O frames.

5.5 Variable Importances

The GBM algorithm automatically calculates variable importances. The model output includes the absolute and relative predictive strength of each feature in the prediction task. To extract the variable importances from the model:

- **In R**: Use `h2o.varimp(air.model)`
- **In Python**: Use `air_model.varimp(return_list=True)`

To view a visualization of the variable importances using the web interface, click the MODEL menu, then select LIST ALL MODELS. Click the INSPECT button next to the model, then select OUTPUT - VARIABLE IMPORTANCES.

5.6 Supported Output

The following algorithm outputs are supported:

- **Regression**: Mean Squared Error (MSE), with an option to output variable importances or a Plain Old Java Object (POJO) model
- **Binary Classification**: Confusion Matrix or Area Under Curve (AUC), with an option to output variable importances or a Java POJO model
- **Classification**: Confusion Matrix (with an option to output variable importances or a Java POJO model)

5.7 Java Models

To access Java code to use to build the current model in Java, click the PREVIEW POJO button at the bottom of the model results. This button generates a POJO model that can be used in a Java application independently of H2O. 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.

For more information on how to use an H2O POJO, refer to the **POJO Quick Start Guide** at https://github.com/h2oai/h2o-3/blob/master/h2o-docs/src/product/howto/POJO_QuickStart.md.

5.8 Grid Search for Model Comparison

To enable grid search capabilities for model tuning, specify sets of values for parameter arguments that will configure certain parameters and then observe the changes in the model behavior. The following example represents a grid search:

Example in R

```
ntrees_opt <- list(5,10,15)
maxdepth_opt <- list(2,3,4)
learnrate_opt <- list(0.1,0.2)
hyper_parameters <- list(ntrees=ntrees_opt, max_depth=
    maxdepth_opt, learn_rate=learnrate_opt)

grid <- h2o.grid("gbm", hyper_params = hyper_
    parameters, y = "IsDepDelayed", x = myX,
    distribution="bernoulli", training_frame = air_
    train.hex, validation_frame = air_valid.hex)
```

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 Model Parameters for parameter descriptions and suggested values.

Example in R

```
# print out all prediction errors and run times of the
    models
grid

# print out the auc for all of the models
grid_models <- lapply(grid@model_ids, function(model_
    id) { model = h2o.getModel(model_id) })
for (i in 1:length(grid_models)) {
  print(sprintf("auc: %f", h2o.auc(grid_models[[i]])))
}
```

5.9 Model Parameters

This section describes the functions of the parameters for GBM.

- `x`: A vector containing the names of the predictors to use while building the GBM model.
- `y`: A character string or index that represents the response variable in the model.
- `training_frame`: An `H2OFrame` object containing the variables in the model.
- `validation_frame`: An `H2OFrame` object containing the validation dataset used to construct confusion matrix. If blank, the training data is used by default.
- `nfolds`: Number of folds for cross-validation.
- `ignore_const_cols`: A boolean indicating if constant columns should be ignored. The default is `TRUE`.
- `ntrees`: A non-negative integer that defines the number of trees. The default is 50.
- `max_depth`: The user-defined tree depth. The default is 5.
- `min_rows`: The minimum number of rows to assign to the terminal nodes. The default is 10.
- `nbins`: For numerical columns (real/int), build a histogram of at least the specified number of bins, then split at the best point The default is 20.
- `nbins_cats`: For categorical columns (enum), build a histogram of the specified number of bins, then split at the best point. Higher values can lead to more overfitting. The default is 1024.
- `seed`: Seed containing random numbers that affects sampling.
- `learn_rate`: An integer that defines the learning rate. The default is 0.1 and the range is 0.0 to 1.0.
- `distribution`: The distribution function options: `AUTO`, `bernoulli`, `multinomial`, `gaussian`, `poisson`, `gamma` or `tweedie`. The default is `AUTO`.
- `score_each_iteration`: A boolean indicating whether to score during each iteration of model training. The default is `FALSE`.

- `fold_assignment`: Cross-validation fold assignment scheme, if `fold_column` is not specified. The following options are supported: `AUTO`, `Random`, or `Modulo`.
- `fold_column`: Column with cross-validation fold index assignment per observation.
- `offset_column`: Specify the offset column. **Note**: Offsets are per-row bias values that are used during model training. For Gaussian distributions, they can be seen as simple corrections to the response (y) column. Instead of learning to predict the response (y-row), the model learns to predict the (row) offset of the response column. For other distributions, the offset corrections are applied in the linearized space before applying the inverse link function to get the actual response values.
- `weights_column`: Specify the weights column. **Note**: Weights are per-row observation weights. This is typically the number of times a row is repeated, but non-integer values are supported as well. During training, rows with higher weights matter more, due to the larger loss function pre-factor.
- `balance_classes`: Balance training data class counts via over or undersampling for imbalanced data. The default is `FALSE`.
- `max_confusion_matrix_size`: Maximum size (number of classes) for confusion matrices to print in the H2O logs. The default is 20.
- `max_hit_ratio_k`: (for multi-class only) Maximum number (top K) of predictions to use for hit ratio computation. To disable, enter `0`. The default is 10.
- `r2_stopping`: Stop making trees when the R^2 metric equals or exceeds this value. The default is 0.999999.
- `build_tree_one_node`: Specify if GBM should be run on one node only; no network overhead but fewer CPUs used. Suitable for small datasets. The default is `FALSE`.
- `tweedie_power`: A numeric specifying the power for the Tweedie function when `distribution = "tweedie"`. The default is 1.5.
- `checkpoint`: Enter a model key associated with a previously-trained model. Use this option to build a new model as a continuation of a previously-generated model.
- `keep_cross_validation_predictions`: Specify whether to keep the predictions of the cross-validation models. The default is `FALSE`.

- `class_sampling_factors`: Desired over/under-sampling ratios per class (in lexicographic order). If not specified, sampling factors will be automatically computed to obtain class balance during training. Requires `balance_classes`.
- `max_after_balance_size`: Maximum relative size of the training data after balancing class counts; can be less than 1.0. The default is 5.
- `nbins_top_level`: For numerical columns (real/int), build a histogram of (at most) this many bins at the root level, then decrease by factor of two per level.
- `model_id`: The unique ID assigned to the generated model. If not specified, an ID is generated automatically.

6 References

Cliff Click. **Building a Distributed GBM on H2O**, 2013. URL http://h2o.ai/blog/2013/10/building-distributed-gbm-h2o/

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

Jane Elith, John R. Leathwick, and Trevor Hastie. **A Working Guide to Boosted Regression Trees**. *Journal of Animal Ecology*, 77(4):802–813, 2008. URL http://onlinelibrary.wiley.com/doi/10.1111/j.1365-2656.2008.01390.x/abstract

Jerome H. Friedman. **Greedy Function Approximation: A Gradient Boosting Machine**. *Annals of Statistics*, 29:1189–1232, 1999. URL http://statweb.stanford.edu/jhf/ftp/trebst.pdf

Jerome Friedman, Trevor Hastie, and Robert Tibshirani. **Discussion of Boosting Papers**. *Annual Statistics*, 32:102–107, 2004. URL http://web.stanford.edu/hastie/Papers/boost_discussion.pdf

Jerome Friedman, Trevor Hastie, and Robert Tibshirani. **Additive Logistic Regression: a Statistical View of Boosting**. *Annals of Statistics*, 28:2000, 1998. URL http://projecteuclid.org/DPubS?service=UI&version=1.0&verb=Display&handle=euclid.aos/1016218223

Jerome Friedman, Trevor Hastie, and Robert Tibshirani. **The Elements of Statistical Learning**. Springer, New York, 2001. URL http://statweb.stanford.edu/tibs/ElemStatLearn/printings/ESLII_print10.pdf

H2O.ai Team. **H2O website**, 2015. URL http://h2o.ai

H2O.ai Team. **H2O documentation**, 2015. URL http://docs.h2o.ai

H2O.ai Team. **H2O GitHub Repository**, 2015. URL https://github.com/h2oai

H2O.ai Team. **H2O Datasets**, 2015. URL http://data.h2o.ai

H2O.ai Team. **H2O JIRA**, 2015. URL https://jira.h2o.ai

H2O.ai Team. **H2Ostream**, 2015. URL https://groups.google.com/d/forum/h2ostream

H2O.ai Team. **H2O R Package Documentation**, 2015. URL http://h2o-release.s3.amazonaws.com/h2o/latest_stable_Rdoc.html

7 Authors

Cliff Click

Cliff Click is the CTO and Co-Founder of H2O, makers of H2O, the open-source math and machine learning engine for Big Data. Cliff wrote his first compiler when he was 15 (Pascal to TRS Z-80!), although Cliffs most famous compiler is the HotSpot Server Compiler (the Sea of Nodes IR). Cliff is invited to speak regularly at industry and academic conferences and has published many papers about HotSpot technology. He holds a PhD in Computer Science from Rice University and about 15 patents.

Jessica Lanford

Jessica is a word hacker and seasoned technical communicator at H2O.ai. She brings our product to life by documenting the many features and functionality of H2O. Having worked for some of the top companies in technology including Dell, AT&T, and Lam Research, she is an expert at translating complex ideas to digestible articles.

Michal Malohlava

Michal is a geek, developer, Java, Linux, programming languages enthusiast developing software for over 10 years. He obtained PhD from the Charles University in Prague in 2012 and post-doc at Purdue University. During his studies he was interested in construction of not only distributed but also embedded and real-time component-based systems using model-driven methods and domain-specific languages. He participated in design and development of various systems including SOFA and Fractal component systems or jPapabench control system.

Viraj Parmar

Viraj is currently an undergraduate at Princeton studying applied mathematics. Prior to joining H2O as a data and math hacker intern, Viraj worked in a research group at the MIT Center for Technology and Design. His interests are in software engineering and large-scale machine learning.

Hank Roark

Hank is a Data Scientist and Hacker at H2O. Hank comes to H2O with a background turning data into products and system solutions and loves helping others find value in their data. He has a deep background in the the application domains of telematics, remote sensing, logistics, manufacturing, agriculture, and the Internet of Things. Hank has an SM from MIT in Engineering and Management and BS Physics from Georgia Tech.

"Smoooooth! - if I have to explain it in one word. H2O made this really easy for R users."

- *Jo-Fai Chow*