Gradient Boosted Machines with H2O

Cliff Click Jessica Lanford Michal Malohlava Viraj Parmar Hank Roark

---

http://h2o.gitbooks.io/gbm-with-h2o/

August 2015: Third Edition

Gradient Boosted Machines with H2O
by Cliff Click, Jessica Lanford, Michal Malohlava, Viraj Parmar, & Hank Roark

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

August 2015: Third Edition

Printed in the United States of America.

Contents

1 Introduction

This document introduces the reader to Gradient Boosted Machines (GBM) with H2O. Examples are written in R and Python. The reader is walked through the installation of H2O, basic GBM concepts, building GBM models in H2O, how to interpret model output, how to make predictions, and various implementation details.

2 What is H2O?

H2O is fast, scalable, open-source machine learning and deep learning for Smarter Applications. With H2O, enterprises like PayPal, Nielsen, Cisco, and others can use all their data without sampling to get accurate predictions faster. Advanced algorithms, like 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, AdTech, 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 was built alongside (and on top of) Hadoop and Spark Clusters 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, time series, k-means clustering, and others. H2O also implements best-in-class algorithms at scale, such as 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 two years, H2O has become a word-of-mouth phenomenon, growing amongst the data community by a hundred-fold, and is now used by 12,000+ users and is deployed using R, Python, Hadoop, and Spark in 2000+ corporations.

Try it out

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

Join the community

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

3 Installation

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

(**Note**: This document was created with H2O version 3.0.1.4.)

3.1 Installation in R

To load a recent H2O package from CRAN, run:

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

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

Alternatively, you can (and should for this tutorial) 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
```

Alternatively, you can (and should for this tutorial) 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

Following 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:

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

3.4 Example code

R 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_Vignette.R`

Python code for the examples in this document can be found here:

`https://github.com/h2oai/h2o-3/blob/master/h2o-docs/src/booklets/v2_2015/source/GBM_Vignette.py`

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`

4 Gradient Boosting Overview

A gradient boosted model 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. 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 human interpretability. The gradient boosting method generalizes tree boosting to minimize these issues.

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
- user-friendly web interface to mirror the model building and scoring process running in R or Python
- grid search for hyperparameter optimization and model selection
- model export in plain Java code for deployment in production environments
- additional parameters for model tuning

4.2 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 `H2OParsedData` 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. If `nfolds >=2`, then `validation_frame` must remain blank. Default is 0.
- `ignore_const_cols`: A boolean indicating if constant columns should be ignored. 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`: Enter `AUTO, bernoulli, multinomial, gaussian, poisson, gamma` or `tweedie` to select the distribution function. The default is `AUTO`.
- `score_each_iteration`: A boolean indicating whether to score during each iteration of model training. 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. Default it 20.
- `max_hit_ratio_k`: (for multi-class only) Maximum number (top K) of predictions to use for hit ratio computation. Use 0 to disable. Default is 10.
- `r2_stopping`: Stop making trees when the R^2 metric equals or exceeds this value. 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. Default is False.
- `tweedie_power`: A numeric specifying the power for the tweedie function when `distribution = "tweedie"`. 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. 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.

4.3 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 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 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 k-classification, H2O builds k-regression trees to represent one classification tree. 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. 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, 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.

4.4 Loss Function

The AdaBoost method builds an additive logistic regression model:

$$F(x) = log \frac{Pr(Y = 1|x)}{Pr(Y = -1|x)} = \sum_{m=1}^{M} \alpha_m f_m(x)$$

by stagewise fitting using the loss function:

$$L(y, F(x)) = exp(-yF(x))$$

4.5 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 in a temporary vector as `Node_ID` 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. A 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 columns. The number of columns 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; therefore, a tree that is five layers deep requires five passes. Each tree level is fully data-parallelized. Each pass is over one layer in the tree, and builds a per-node histogram in the MR calls. As each pass analyzes a tree level, H2O then decides how to build the next level. H2O reassigns rows to new levels in another pass by merging the two passes and builds a histogram for each node. Each per-level histogram is done in parallel.

Scoring and building is done in one pass. Each row is tested against the decision from the previous pass, assigned to a new leaf, and a histogram is built on that leaf. 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.

High max-bin sizes slow down generation of deep trees, but this is not a problem for GBM tree depths as GBM typically uses very shallow trees. (TODO: Check, this probably isn't true anymore) Bin limits over 20 have been shown to provide very little improvement on the quality of the GBM model.

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.

4.6 Treatment of factors

When the specified GBM model includes factors, those factors are analyzed by assigning an integer to each distinct factor level, and then binning the ordered integers according to the user-specified number of bins (N bins). Split points are determined by considering the end points of each bin and the one-versus-many split for each bin.

For example, if the factor is split into five bins, H2O orders the bins by bin number, then considers the split between the first and second bin, then the second and third, then the third and fourth, and the fourth and fifth. Additionally, the split that results from splitting the first bin from the other four and all analogous splits for the other four bins are considered. To specify a model that considers all factors individually, set the value for N bins equal to the number of factor levels. This can be done for over 1024 levels (the maximum number of levels that can be handled in R), though this increases the time required to fully generate a model.

Increasing the number of bins is less useful for covering factor columns, but is more important for the one-versus-many approach. The "split-by-a-numerical-value" is basically a random split of the factors, so the number of bins is less important. Top-level tree splits (shallow splits) use the maximum allotment as their bin size, so the top split uses 1024 bins, the next level in the tree uses 512 bins, and so on.

Factors for binary classification have a third (and optimal) choice: to split all bins (and factors within those bins) that have a mean of less than 0.5 one way, and the rest of the bins and factors the other way, creating an arbitrary set of factors with a known optimal split. This is represented as a bitset in the embedded Java model. Therefore, factor columns with less than the limit ($\sim$ 1024) always get an optimal split for categorical problems.

For categorical problems with N possible values, the split candidate is determined by the formula $2^{N-1}-1$. For binary classification and regression problems, the number of split candidates is reduced to $N-1$ by

sorting the categorical feature values by label average.

4.7 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 $= \#\text{columns} \cdot N \cdot K \cdot 2^J$. 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

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 Loading data

Loading a dataset in R or Python 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`.

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 weather.hex <- h2o.uploadFile(h2o_server, path  = "weather.csv", header
    = TRUE, sep = ",", destination_frame = "weather.hex")
2
3 # To see a brief summary of the data, run the following command.
4 summary(weather.hex)
```

Example in Python

```
weather_hex = h2o.import_file(path = "weather.csv")

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

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.

Example in R

```
# Load the data and prepare for modeling
airlines.hex <- h2o.uploadFile(h2o_server, 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, 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 R users have access to an intuitive web interface for H2O, Flow, to mirror the model building process in R. After loading data or training a model in R, point your browser to your IP address and port number (e.g., localhost:12345) to launch the web interface. From here, you can click on ADMIN > JOBS to view

specific details about your model. You can also click on DATA > LIST ALL FRAMES to view all current H2O frames.

5.5 Variable importances

GBM algorithm will automatically calculate variable importances. The display includes the the absolute and relative predictive strength of each feature in the prediction task. From R, use the command `h2o.varimp(air.model)` to extract the variable importances from the model; from Python, use the command You can also view a visualization of the variable importances on the web interface.

5.6 Supported Output

The following algorithm outputs are supported:

- **Regression**: Mean Squared Error (MSE), with an option to output variable importances or a Java 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 model

To access Java (POJO) code to use to build the current model in Java, click the PREVIEW POJO button at the bottom of the model results. 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.

5.8 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:

Example in R

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

```
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 the beginning section 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]])))
}
```

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

Click, Cliff and SriSatish Ambati. **"Cliff Click Explains GBM at Netflix October 10 2013"** http://www.slideshare.net/0xdata/cliff-click-explains-gbm SlideShare (2013).

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.