Gradient Boosted Machines with H2O's R Package

August 2014

Contents

1 Introduction

This vignette presents the gradient boosted machine (GBM) framework in the H2O package. Further documentation on H2O's system and algorithms can be found at the 0xdata website and the R package manual. The full datasets and code of this vignette can be found at the H2O Github. This introductory section provides instructions on getting H2O started, followed by a brief overview of gradient boosting.

1.1 Installation

You can load the latest CRAN H2O package by running

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

Alternatively, you can (and should for this tutorial) download the latest H2O build by following the "Install in R" instructions in the H2O download table. Open your R Console and run the following to install the latest H2O build in R:

```
# The following two commands remove any previously installed H2O packages for R.
if ("package:h2o" %in% search()) { detach("package:h2o", unload=TRUE) }

if ("h2o" %in% rownames(installed.packages())) { remove.packages("h2o") }

# Next, we download, install and initialize the H2O package for R (filling in the
*'s with the matching digits in the download table)
install.packages("h2o", repos=(c("http://s3.amazonaws.com/h2o-release/h2o/master/
****/R", getOption("repos"))))

library(h2o)
```

If you are operating on a single node then initialize H2O with

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

With this command, the H2O R module will start an instance of H2O automatically at localhost:54321. Alternatively, to specify a connection with an existing H2O cluster node (other than localhost at port 54321) you must explicitly state the IP address and port number in the `h2o.init()` call. An example is given below, but do not directly paste; you should specify the IP and port number appropriate to your specific environment.

```
h2o_cluster = h2o.init(ip = "192.555.1.123", port = 12345, startH2O = FALSE)
```

An automatic demo is available to see h2o.gbm at work. Run the following command to observe an example classification model built through H2O's GBM.

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

1.2 Support

Users of the H2O package may submit general enquiries and bug reports to the 0xdata support address. Alternatively, specific bugs or issues may be filed to the 0xdata JIRA.

1.3 Gradient Boosting overview

1.3.1 Summary of features

H2O's GBM functionalities include:

- supervised learning for regression and classification tasks
- distributed and parallelized computation to be run 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

1.3.2 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, on the other hand, 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).

GBM for classification

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

In the above algorithm, the index m keeps track 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 0xdata blog.

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

1.3.3 Key parameters

In the above example, an important user-specified value is N, which represents the number of bins data are partitioned into before the tree's best split point is determined. Split points are determined by considering as the end points of each bin, and the one versus many split for each bin. High values of N may be specified if users wish to model all factors individually, but this will slow down the modeling process. For shallow trees, we keep the total count of bins arcoss all splits at 1024 – so a top-level split will use 1024, but a 2nd level split will usew 512 bins, and so forth. This value is then maxed with the input bin count.

Another important parameters that users specify is the size J of the trees, which must be controlled in order to avoid overfitting. Increasing J allows for larger variable interaction effects, so intuition 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 we have that Cost = #columns $\cdot N \cdot K \cdot 2^J$. In general, however, lower values have also been found to have the highest performance. Models with $4 \leq J \leq 8$ and a larger number of trees M reflect this sound rule of thumb. Later, we will see how grid search models can allow for tuning of these parameters in the model selection process.

In addition users can specify the shrinkage constant, which controls the learning rate of the model and is actually a form of regularization. Shrinkage modifies the algorithms update of $f_{km}(x)$ with instead a 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 training error, assuming that M is held fixed, but in general ν and M are inversely related when the error is held fixed. Despite the greater training error with small values of ν, however, it has been found that very small values ($\nu < 0.1$) lead to better generalization and hence performance on test data.

2 Use case: Classification with Airline data

2.1 Airline dataset overview

The Airline dataset can be downloaded here. Remember to save the .csv file to your working directory. Before running the Airline demo we first review how to load data with H2O.

2.1.1 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 an example, we use a toy weather dataset which can be downloaded here. First load the data to your current working directory in your R Console (do this henceforth for 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 quick summary of the data, run the following command.

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

2.2 Performing a trial run

Returning to the Airline dataset demo, we first load the dataset with H2O and select which variables we wish to use to predict a chosen response. For example, we may want to 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 we 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)
```

The model is built with only 100 trees since it is meant just as a trial run. In this trial run we also did not specify a validation set; this defaults the model evaluation to the entire training set but another option is to use n-fold validation by specifying, for example, `nfolds=5`.

2.2.1 Extracting and handling the results

We can extract the parameters of our 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 latter command returns the trained model's training and validation error.

Once we have a satisfactory model, the `h2o.predict()` command can be used 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)
```

2.3 Web interface

H2O R users have access to a slick web interface to mirror the model building process in R. After loading data or training a model in R, point your browser to your IP address+port number (e.g., localhost:12345) to launch the web interface. From here you can click on Admin > Jobs to view your specific model details. You can also click on Data > View All to view and keep track of your datasets in current use.

2.3.1 Variable importances

One useful feature is the variable importances option, which can be enabled with the additional argument `importance=TRUE`. This features allows us to view the absolute and relative predictive strength of each feature in the prediction task. From R, you can access these strengths with the command `air.model@model$varimp`. You can also view a visualization of the variable importances on the web interface.

2.3.2 Java model

Another important feature of the web interface is the Java (POJO) model, accessible from the Java model button in the top right of a model summary page. This button allows access to Java code which, when called from a main method in a Java program, builds the model at hand. When the model is small enough, the java code for the model will be made available to inspect from within the GUI, larger models can be inspected after users have downloaded the model.

To download the model open the terminal window, create a directory where the model will be saved, set the new directory as the working directory and follow the curl and java compile commands displayed in the instructions at the top of the java model.

2.4 Grid search for model comparison

H2O supports grid search capabilities for model tuning by allowing users to tweak certain parameters and observe changes in model behavior. This is done by specifying sets of values for parameter arguments. For example, below 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))
```

Here we specified 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, as we inspect and compare our trained models after the grid search process is complete. To decide how and when to choose different parameter configurations in a grid search, see 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
```

3 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.

4 References

Dietterich, Thomas G, and Eun Bae Kong. Machine Learning Bias, Statistical Bias, and Statistical Variance of Decision Tree Algorithms. ML-95 255 (1995).

Elith, Jane, John R Leathwick, and Trevor Hastie. A Working Guide to Boosted Regression Trees. Journal of Animal Ecology 77.4 (2008): 802-813

Friedman, Jerome H. Greedy Function Approximation: A Gradient Boosting Machine. Annals of Statistics (2001): 1189-1232.

Friedman, Jerome, Trevor Hastie, Saharon Rosset, Robert Tibshirani, and Ji Zhu. Discussion of Boosting Papers. 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). The Annals of Statistics 28.2 (2000): 337-407

Hastie, Trevor, Robert Tibshirani, and J Jerome H Friedman. The Elements of Statistical Learning. Vol.1. N.p., page 339: Springer New York, 2001.

Niu, Feng, et al. Hogwild!: A lock-free approach to parallelizing stochastic gradient descent. Advances in Neural Information Processing Systems 24 (2011): 693-701. (algorithm implemented is on p.5)