h2oml — Introduction to commands for Stata integration with H2O machine learning

Description

This entry describes commands for performing predictive analysis using H2O machine learning methods, specifically ensemble decision tree methods, in Stata. H2O is a scalable and distributed machine learning and predictive analytics platform that allows you to perform data analysis and machine learning. It provides parallelized implementations of many widely used supervised and unsupervised machine learning methods. For more details, see [H2OML] **H2O setup**, [P] **H2O intro**, and https://www.stata.com/h2o/h2o18/h2o_intro.html#what-is-h2o. For a software-free introduction to machine learning, see [H2OML] **Intro**.

Supervised learning

h2oml gbm	gradient boosting machine
`h2oml gbregress`	gradient boosting regression
`h2oml gbbinclass`	gradient boosting binary classification
`h2oml gbmulticlass`	gradient boosting multiclass classification
h2oml rf	random forest
`h2oml rfregress`	random forest regression
`h2oml rfbinclass`	random forest binary classification
`h2oml rfmulticlass`	random forest multiclass classification

Estimation results and postestimation frame

`h2omlest store`	catalog H2O estimation results
`h2omlpostestframe`	specify frame for postestimation analysis

Tuning and estimation summaries

`h2omlestat metrics`	display performance metrics
`h2omlgof`	goodness of fit for machine learning methods
`h2omlestat cvsummary`	display cross-validation summary
`h2omlestat gridsummary`	display grid-search summary
`h2omlexplore`	explore models after grid search
`h2omlselect`	select model after grid search
`h2omlgraph scorehistory`	produce score history plot

Performance after binary classification

`h2omlestat threshmetric`	display threshold-based metrics
`h2omlestat confmatrix`	display confusion matrix
`h2omlgraph prcurve`	produce precision–recall curve plot
`h2omlgraph roc`	produce ROC curve plot

Performance after multiclass classification

`h2omlestat aucmulticlass`	display AUC and AUCPR summary
`h2omlestat confmatrix`	display confusion matrix
`h2omlestat hitratio`	display hit-ratio table

Prediction

`h2omlpredict`	prediction of continuous responses, probabilities, and classes

Machine learning explainability

`h2omlgraph varimp`	produce variable importance plot
`h2omlgraph pdp`	produce partial dependence plot
`h2omlgraph ice`	produce individual conditional expectation plot
`h2omlgraph shapvalues`	produce SHAP values plot for individual observations after regression and binary classification
`h2omlgraph shapsummary`	produce SHAP beeswarm plot after regression and binary classification

Save decision tree

`h2omltree`	save decision tree DOT file and display rule set

Remarks and examples

This entry describes Stata commands to perform predictive analysis using H2O machine learning ensemble decision tree methods.

Remarks and examples are presented under the following headings:

Brief overview

The h2oml suite of Stata commands provides end-to-end support for H2O machine learning analysis using ensemble decision tree methods. In addition to h2oml, the _h2oframe command provides several key subcommands that connect Stata to an H2O cluster, import a Stata dataset into an H2O frame, and provide various H2O data management; see [H2OML] **H2O setup**.

h2oml *gbm* and h2oml *rf* provide the suite of estimation commands that implement gradient boosting and random forest regression, binary classification, and multiclass classification. h2oml gbregress and h2oml rfregress perform respective gradient boosting and random forest regressions for continuous and count responses, h2oml gbbinclass and h2oml rfbinclass perform gradient boosting and random forest classifications for binary responses, and h2oml gbmulticlass and h2oml rfmulticlass perform gradient boosting and random forest classifications for categorical responses (with more than two categories).

All commands provide the validframe() and cv() options to specify a validation frame and to perform cross-validation to control for overfitting, the tune() and stop() options to tune hyperparameters and stop early for better model performance, the h2orseed() option to reproduce results, and many more. Many commands also offer specialized options such as the loss() option of h2oml gbregress, which specifies various loss functions, including quantile, Huber, and Tweedie. See [H2OML] ***h2oml gbm*** and [H2OML] ***h2oml rf*** for details.

After estimation, the h2omlest suite of commands can be used to manage estimation results. For instance, h2omlest store can be used to store the current estimation results for later use.

Several postestimation commands are available to obtain tuning and estimation summaries. For instance, h2omlestat gridsummary is useful to view the results after tuning and select an alternative model that is more parsimonious. And h2omlgraph scorehistory can be used to display various validation curves to help monitor overfitting.

For binary and multiclass classifications, several commands can be used to explore model performance such as the h2omlestat confmatrix command, which displays the confusion matrix. Additionally, h2omlgraph prcurve and h2omlgraph roc can be used to plot precision–recall and receiver operating characteristic (ROC) curves after binary classification, and h2omlestat hitratio can be used to produce a hit-ratio table after multiclass classification.

The ultimate goal of machine learning is to obtain accurate prediction of the response on the new data. To achieve this goal, the model predictive performance is often evaluated by using an external, testing dataset. The h2omlpostestframe command provides a convenient way to specify the desired testing frame to be used in all subsequent postestimation analyses.

Depending on the estimation method, regression or classification, the h2omlpredict command produces predictions of continuous and count responses or class probabilities and classes.

Machine learning methods are often treated as a black box, meaning that little attempt is made to understand the obtained predictions. To rectify this, h2oml provides several postestimation commands to help explain predictions. The h2omlgraph varimp command can be used to assess the overall importance of predictors in the model, whereas the h2omlgraph shapvalues and h2omlgraph shapsummary commands can be used to explore the impact of predictors on individual predictions.

Finally, the h2omltree command can be used to save a specific decision tree in a DOT file and plot it by using the open source software Graphviz; see [H2OML] **DOT extension**.

For more details about postestimation commands, see [H2OML] **h2oml postestimation**.

h2oml in a nutshell

In the previous section, we briefly described the functionality of the `h2oml` command. Here we will provide a quick overview of some of the more common usages of this command in practice.

As we mentioned earlier, machine learning is primarily used to develop a model that accurately predicts a response of interest on the new data. In practice, several general steps are often performed to build such a model.

At the beginning of the analysis, the data are often split into training data used for estimation and validation data used for evaluating the model performance. Additionally, external testing data are also available for assessing the model final predictive performance and comparing it with other models that use a different machine learning method such as gradient boosting machine (GBM) or random forest. For each method, models with different sets of hyperparameters are evaluated using a validation dataset (or cross-validation), and the best model is chosen. The chosen models are further evaluated based on their predictive performance on the testing data, and the final model is selected for later prediction on the future new data.

Below, we describe several `h2oml` commands that can be used to perform the above steps.

Setup. To use the `h2oml` command, we must first initialize an H2O cluster and import our data to an H2O frame; see *Prepare your data for* H2O *machine learning in Stata* and [H2OML] **H2O setup**. Here we load the current Stata dataset into the H2O `data` frame and make it the current H2O frame.

```
. h2o init
. _h2oframe put, into(data)
. _h2oframe change data
```

Alternatively, we could replace the last two commands with `_h2oframe put, into(data) current` to put the dataset into an H2O frame and make this frame current in a single step.

Next we split the `data` frame into training and validation with, say, 80% of observations in the training sample. We also specify the random-number seed for reproducibility and make the `train` frame be the current H2O frame for estimation.

```
. _h2oframe split data, into(train valid) split(0.8 0.2) rseed(19)
. _h2oframe change train
```

Depending on the type of a response and the desired machine learning method, we can choose one of the six `h2oml` commands to perform estimation: `h2oml gbregress`, `h2oml gbbinclass`, `h2oml gbmulticlass`, `h2oml rfregress`, `h2oml rfbinclass`, and `h2oml rfmulticlass`.

Reference or baseline model. Suppose we have a binary response and we want to use GBM. We can start with a simple reference model with default hyperparameters:

```
. h2oml gbbinclass response predictors, h2orseed(19) validframe(valid)
```

We specified the `h2orseed(19)` option to ensure H2O reproducibility; see [H2OML] **H2O reproducibility**.

If we do not have sufficient observations to split the data into training and validation, we can use cross-validation instead such as a 3-fold cross-validation with the default random splitting of the data below:

```
. h2oml gbbinclass response predictors, h2orseed(19) cv(3)
```

We store the current estimation results to use as a benchmark later.

```
. h2omlest store gbm_ref
```

User-specified hyperparameters and tuning. Next we can explore models with values of hyperparameters other than the default ones. For instance, we can specify 200 trees instead of the default 50 and a 0.2 learning rate instead of the default 0.1. And we can specify different values for any of the other nine hyperparameters; see *Options* in [H2OML] ***h2oml gbm***.

```
. h2oml gbbinclass response predictors, h2orseed(19) cv(3)
> ntrees(200) lrate(0.2) ...
```

We store this model as

```
. h2omlest store gbm_user
```

In practice, it is difficult to know the actual hyperparameter values that provide the best model performance, so an iterative procedure known as hyperparameter tuning is used to explore different ranges of various hyperparameters to select the best set of values. To incorporate tuning, the `h2oml` estimation commands allow you to specify the ranges (*numlist*) in options for hyperparameters and provide the `tune()` option to control the tuning procedure.

Which hyperparameters should be tuned and what ranges should be explored will be specific to each application. Here, for illustration purposes and continuing with our example, we will tune the number of trees and the learning rate:

```
. h2oml gbbinclass response predictors, h2omlrseed(19) cv(3)
> ntrees(20(10)200) lrate(0.1(0.1)1)
```

We store this tuned model as

```
. h2omlest store gbm_tuned
```

If desired, we can change the default tuning metric (from log loss to, say, accuracy) and grid-search method (from Cartesian to random) as well as specify other suboptions in the `tune()` option:

```
. h2oml gbbinclass response predictors, h2omlrseed(19) cv(3)
> ntrees(20(10)200) lrate(0.1(0.1)1)
> tune(metric(accuracy) grid(random) ...)
```

Checking for overfitting or underfitting. Before we proceed with model selection, we can check for model overfitting or underfitting. We can use the `h2omlgraph scorehistory` command to plot the metric values against the number of trees to compare the training and validation or cross-validation curves:

```
. h2omlgraph scorehistory
```

The number of trees at which the two curves start noticeably diverging provides a tradeoff between underfitting and overfitting.

Because we performed cross-validation, it is also useful to evaluate its performance. We can check the variability of the metric values across the folds with

```
. h2omlestat cvsummary
```

High variation may indicate overfitting.

Our current model is `gbm_tuned`, but we can repeat the above steps for the other two models by first using the `h2omlest restore` command to restore their estimation results.

Selecting the "best" model. Our current `gbm_tuned` model uses the hyperparameter values that resulted in the smallest value of the default log loss metric. We can evaluate alternative models that may be more parsimonious and thus may run faster:

```
. h2omlestat gridsummary
```

We can also explore the performance of additional metrics for different models before deciding on a model. For instance, we can explore the top 10 models:

```
. h2omlexplore id = 1(1)10
```

If we find an alternative model that we think is best, we can switch to it by using

```
. h2omlselect id = #
```

where # is an index of the corresponding model from `h2omlestat gridsummary`.

To select between all the considered models with different hyperparameters such as `gbm_tuned` and `gbm_user`, we select the one with the most optimal metric value, which is reported in the output of the `h2oml` estimation commands. We can also use

```
. h2omlestat metrics
```

to report the performance metrics for the current estimation model.

And we can compare different metrics side by side for all models more easily by using

```
. h2omlgof gbm_tuned gbm_user gbm_ref
```

Evaluate predictive performance and compare different methods. Predictive performance of a model is typically evaluated on an external testing dataset. The `h2omlpostestframe` command provides a convenient way of specifying a testing frame for all postestimation analyses:

```
. h2omlpostestframe test
```

Here `test` is our H2O testing frame. This command does not physically change the current frame from `train` to `test`. It instead specifies that all relevant postestimation commands use the `test` frame in the computations instead of their specific default frames, which may be training, validation, or cross-validation depending on the estimation.

After binary or multiclass classification, we can evaluate model predictive performance by using the confusion matrix:

```
. h2omlestat confmatrix
```

After binary classification, we can also explore thresholds that are optimal for various metrics

```
. h2omlestat threshmetric
```

Here we chose to use a GBM method. We can also consider using a random forest method. We would repeat all the above steps but now using the `rfbinclass` command for estimation to select the best random forest model, say `rf_tuned`. We would then use the above commands to compare the predictive performances of the two models or use

```
. h2omlgof gbm_tuned rf_tuned
```

to compare different performance metrics side by side. We can compare different methods using precision–recall and ROC curves:

```
. h2omlgraph prcurve, models(gbm_tuned rf_tuned)
. h2omlgraph roc, models(gbm_tuned rf_tuned)
```

Obtain predictions. Once the best model is chosen, we can use it to compute predictions. Depending on the research question, we can compute predictions for an entirely new dataset, or we can use the original data. Here we obtain predictions for our original `data` frame.

```
. _h2oframe change data
. h2omlpredict
```

Explain predictions. The `h2oml` suite provides several commands for explaining predictions. We can evaluate overall predictors' importance that quantifies the effect of each predictor on the model's predictions:

```
. h2omlgraph varimp
```

We can also use the partial dependence plot (PDP) and the individual conditional expectation (ICE) plot to visually explore predictor dependence on the response:

```
. h2omlgraph pdp  predictors
. h2omlgraph ice  predictor
```

And, after regression and binary classification, we can use Shapley additive explanations (SHAP) values to explore predictor contributions to the prediction of the response:

```
. h2omlgraph shapvalues
. h2omlgraph shapsummary
```

Tour of machine learning commands

In this section, we illustrate the usage of the `h2oml` command with applications to several real-world datasets. We start by showing how to start an H2O cluster and convert your Stata dataset into an H2O frame. We then illustrate the basic steps for training machine learning methods and provide predictions for binary classification and for regression. We also explore the effect of categorical predictors on the performance of ensemble decision tree methods and demonstrate how to use these methods to detect important predictors. We also show a quick analysis of a count response by using a gradient boosting Poisson regression.

Examples are presented under the following headings:

Prepare your data for H2O machine learning in Stata

Before using any of the H2O machine learning methods in Stata, you need to connect to or initialize an H2O server by using the h2o init command. The command first checks whether an H2O cluster is already running on the local machine and uses that cluster if so; otherwise, it attempts to start a new cluster. For details, see [H2OML] **H2O setup**.

We first use the h2o init command to start an H2O cluster.

```
. h2o init
```

Suppose we have an external data.csv file saved in Stata's current directory. We can import it as an H2O frame by typing

```
. _h2oframe import data.csv, into(data)
```

or if we already have our data loaded into Stata, we can store it as an H2O frame by typing

```
. _h2oframe put, into(data)
```

In the above, we put our data into the H2O cluster as an H2O frame and called it data. To be able to work with the data frame, we need to change it to be the current working frame:

```
. _h2oframe change data
```

Before starting any H2O analysis, we recommend that you describe the data to ensure that the H2O variable types are as expected. This is important because the implementation of H2O machine learning methods can vary depending on the types of the response and predictors.

```
. _h2oframe describe
```

Suppose our data have two variables: y and x. To run a regression for y on x using GBM with default settings, we can now type

```
. h2oml gbregress y x
```

Or we can use random forest with default settings by typing

```
. h2oml rfregress y x
```

After estimation, we can use any postestimation command from [H2OML] **h2oml postestimation**.

End-to-end binary classification analysis

In this section, we provide an end-to-end analysis for a binary classification problem using gradient boosting binary classification. The examples comprise tuning, performance analysis, and prediction explainability.

▷ Example 1: Data setup

Consider data from a fictional company, Telco, that provides home phone and internet services in California. The data have been made available by IBM. We want to build a predictive model to predict the behavior of a customer who is more likely to churn. churn.dta contains 7,043 observations and 26 variables. The binary response churn indicates whether a customer left within the last month or is still using Telco's services. The predictors include customers' demographic information such as gender and age, customers' account information such as payment period and duration of services, customers' service types such as whether a customer signed up for internet, phone, device protection, etc.

The goal of this example is to build a predictive model that will predict the behavior of a customer who is more likely to churn or retain the company's services.

As we described in *Prepare your data for H2O machine learning in Stata*, we start by reading the dataset as an H2O frame. We then describe the frame to make sure that variables (H2O columns) have the intended data types by using the `_h2oframe describe` command. Recall that `h2o init` initiates an H2O cluster and `_h2oframe put` loads the current Stata dataset into an H2O frame. For details, see [H2OML] **H2O setup**.

```
. use https://www.stata-press.com/data/r19/churn
(Telco customer churn data)

. h2o init
  (output omitted)

. _h2oframe put, into(churn)

Progress (%): 0 100

. _h2oframe change churn

. _h2oframe describe

          Rows:      7043
          Cols:        26

------------------------------------------------------------------------------
Column          Type          Missing      Zeros       +Inf       -Inf  Cardinality
------------------------------------------------------------------------------

zipcode         int                 0          0          0          0
latitude        real                0          0          0          0
longitude       real                0          0          0          0
tenuremonths    int                 0         11          0          0
monthlycharges  real                0          0          0          0
totalcharges    real               11          0          0          0
country         enum                0       7043          0          0            1
state           enum                0       7043          0          0            1
city            enum                0          4          0          0         1129
gender          enum                0       3488          0          0            2
seniorcitizen   enum                0       5901          0          0            2
partner         enum                0       3641          0          0            2
dependents      enum                0       5416          0          0            2
phoneservice    enum                0        682          0          0            2
multiplelines   enum                0       3390          0          0            3
internetserv    enum                0       2421          0          0            3
onlinesecurity  enum                0       3498          0          0            3
onlinebackup    enum                0       3088          0          0            3
deviceprotect   enum                0       3095          0          0            3
techsupport     enum                0       3473          0          0            3
streamtv        enum                0       2810          0          0            3
streammovie     enum                0       2785          0          0            3
contract        enum                0       3875          0          0            3
paperlessbill   enum                0       2872          0          0            2
paymethod       enum                0       1544          0          0            4
churn           enum                0       5174          0          0            2
------------------------------------------------------------------------------
```

For definitions of data types in H2O, see https:/www.stata.com/h2o/h2oframe_intro.html. Specifically, `enum` refers to categorical or factor columns in an H2O frame, `real` to numeric columns with `float` or `double` values, and `int` to numeric columns with integer values. For example, here `churn` has the expected type `enum`. If the data types are incorrect, `_h2oframe` provides commands to convert an H2O frame column to the desired data type; see https://www.stata.com/h2o/h2oframe.html. You may notice that the predictor `totalcharges` has 11 missing values. As we discussed in *Decision trees* of [H2OML] **Intro**, tree-based methods naturally handle missing values.

Next we split our data into training and testing frames with 80% of observations in the training sample. We will use cross-validation on training data during estimation to control for overfitting.

```
. _h2oframe split churn, into(train test) split(0.8 0.2) rseed(19)

. _h2oframe change train
```

◁

▷ Example 2: Reference binary classification using GBM

As we discussed in *Model selection in machine learning* of [H2OML] **Intro**, the analysis should start by defining a baseline or reference performance.

For classification problems, it is recommended to first check whether the dataset is imbalanced.

```
. tabulate churn

   Churning │
     status │      Freq.     Percent        Cum.
────────────┼───────────────────────────────────
         No │      5,174       73.46       73.46
        Yes │      1,869       26.54      100.00
────────────┼───────────────────────────────────
      Total │      7,043      100.00
```

Our dataset suffers from imbalance. Therefore, we will use the stratification method for cross-validation to ensure that the cross-validation samples maintain the same data imbalance. Following the literature on measuring performance for imbalanced data (Davis and Goadrich 2006), we will use area under the precision–recall curve (AUCPR) as a performance metric in our analysis.

Next, for convenience, let's create a global macro, `predictors`, in Stata to store the names of predictors.

```
. global predictors latitude longitude tenuremonths monthlycharges totalcharges
> gender seniorcitizen partner dependents phoneservice multiplelines
> internetserv onlinesecurity onlinebackup streamtv techsupport streammovie
> contract paperlessbill paymethod deviceprotect
```

As a reference model, we fit a GBM model with a 3-fold stratified cross-validation and default values for other settings. We specify the `h2orseed(19)` option for reproducibility; see [H2OML] **H2O reproducibility**.

```
. h2oml gbbinclass churn $predictors, h2orseed(19) cv(3, stratify)
Progress (%): 0 42.5 87.0 100

Gradient boosting binary classification using H2O

Response: churn
Loss:     Bernoulli
Frame:                                      Number of observations:
  Training: train                                      Training =  5,643
                                               Cross-validation =  5,643
Cross-validation: Stratify                  Number of folds     =      3

Model parameters

Number of trees      =  50                  Learning rate       =     .1
              actual =  50                  Learning rate decay =      1
Tree depth:                                 Pred. sampling rate =      1
           Input max =   5                  Sampling rate       =      1
                 min =   5                  No. of bins cat.    =  1,024
                 avg = 5.0                  No. of bins root    =  1,024
                 max =   5                  No. of bins cont.   =     20
Min. obs. leaf split =  10                  Min. split thresh.  = .00001

Metric summary
```

Metric	Training	Cross-validation
Log loss	.3293387	.411338
Mean class error	.1603572	.2338787
AUC	.9163226	.8500772
AUCPR	.8023966	.6584908
Gini coefficient	.8326452	.7001545
MSE	.1034999	.1350446
RMSE	.321714	.3674841

For detailed interpretation of the output, see example 1 of [H2OML] ***h2oml gbm***.

Although we are mainly interested in cross-validation metrics, we still need to examine the training metrics to make sure that we slightly overfit the training data to avoid underfitting. The latter can be checked by exploring the difference between training and cross-validation metrics, which should be positive for the AUCPR metric. However, if the difference between the validation and training metrics is large, it indicates that the model is too tailored to the training data and may not generalize well to new data. In the literature, there is no clear recommendation on how large the difference between training and validation metrics should be to indicate severe overfitting. Each case should be evaluated individually and with caution. For details, see Valdenegro-Toro and Sabatelli (2023). In our example, the positive difference between the training and cross-validation AUCPR values suggests that our model does overfit the training data. The cross-validation AUCPR for the reference model is approximately 0.658.

We store the reference estimation results for later comparison using the `h2omlest store` command.

```
. h2omlest store gbm_default
```

It is helpful to assess the variance of each metric over the folds to ensure that the model performance does not depend on the specific split of the data. Large variation of the cross-validation metrics over the folds may lead to poor generalization of the model to new data. In such cases, it is recommended to adjust the number of folds or examine the data to identify the sources of variability. We can use `h2omlestat cvsummary` to display cross-validation summary.

```
. h2omlestat cvsummary
Cross-validation summary using H2O
────────────────────────────────────────────────────────────────────────────────
                   Metric │      Mean   Std. dev.      Fold 1      Fold 2
──────────────────────────┼─────────────────────────────────────────────────────
                 Log loss │  .4113427    .0038855    .4085804    .4157856
                       F1 │  .6401071    .0044256    .6358885    .6397188
                       F2 │  .6954293    .0055981    .6891994    .6970509
                     F0.5 │  .5929428    .0039657    .5902329     .591101
                 Accuracy │  .7806169    .0012531    .7793031    .7817988
                Precision │  .5651822    .0039084    .5632716    .5625966
                   Recall │  .7379531    .0069124         .73    .7413442
              Specificity │  .7959458    .0011321    .7969871    .7961095
        Misclassification │  .2193831    .0012531    .2206969    .2182012
         Mean class error │  .2330506    .0029933    .2365065    .2312731
         Max. class error │  .2620469    .0069124         .27    .2586558
      Mean class accuracy │  .7669494    .0029933    .7634935    .7687268
  Misclassification count │  412.6667    4.618802         418         410
                      AUC │  .8505131    .0040418    .8526636    .8458507
                    AUCPR │  .6597555    .0045358    .6628664     .654551
                      MSE │  .1350454    .0017733    .1340862    .1370917
                     RMSE │  .3674799    .0024083    .3661779     .370259
────────────────────────────────────────────────────────────────────────────────

──────────────────────────────────────
                   Metric │    Fold 3
──────────────────────────┼───────────
                 Log loss │  .4096621
                       F1 │  .6447141
                       F2 │  .7000377
                     F0.5 │  .5974944
                 Accuracy │  .7807487
                Precision │  .5696784
                   Recall │   .742515
              Specificity │  .7947407
        Misclassification │  .2192513
         Mean class error │  .2313722
         Max. class error │   .257485
      Mean class accuracy │  .7686278
  Misclassification count │       410
                      AUC │  .8530251
                    AUCPR │  .6618491
                      MSE │  .1339582
                     RMSE │  .3660029
──────────────────────────────────────
```

In our example, the variation of the cross-validation metrics across folds, that is, AUCPR, is small. The mean value of the cross-validation AUCPR is around 0.660, which is slightly different from the cross-validation AUCPR of 0.658 reported by `h2oml gbbinclass`. This difference is expected because of how the two commands compute cross-validation metrics. `h2omlestat cvsummary` computes metrics separately for each fold and reports their average value, whereas `h2oml gbbinclass` combines all folds into one and computes a single AUCPR value.

◁

▷ Example 3: Model selection and hyperparameter tuning

Hyperparameters, such as the number of trees and learning rate, control the performance of a machine learning model. Choosing the "right" hyperparameters can substantively improve both the model performance and its ability to be generalized to new data. Poorly selected hyperparameters, on the other hand, can lead to underfitting or overfitting. The process of selecting hyperparameters to achieve optimal model performance is known as hyperparameter tuning.

In example 5 of [H2OML] ***h2oml gbm***, we demonstrated the detailed steps of hyperparameter tuning for this example. Here we use the final selected model:

```
. h2oml gbbinclass churn $predictors, h2orseed(19) cv(3, stratify)
> ntrees(100) lrate(0.05) predsamprate(0.15)
Progress (%): 0 36.0 71.2 89.4 100

Gradient boosting binary classification using H2O

Response: churn
Loss:     Bernoulli
Frame:                                 Number of observations:
  Training: train                                 Training =  5,643
                                          Cross-validation =  5,643
Cross-validation: Stratify             Number of folds     =      3

Model parameters

Number of trees       = 100            Learning rate       =    .05
              actual  = 100            Learning rate decay =      1
Tree depth:                            Pred. sampling rate =    .15
           Input max  =   5            Sampling rate       =      1
                 min  =   5            No. of bins cat.    =  1,024
                 avg  = 5.0            No. of bins root    =  1,024
                 max  =   5            No. of bins cont.   =     20
Min. obs. leaf split  =  10            Min. split thresh.  = .00001

Metric summary

                                          Cross-
               Metric    Training   validation

             Log loss     .3531063    .4026141
     Mean class error     .1784776    .2313897
                  AUC     .8992847    .8565935
                AUCPR     .7610732     .673929
     Gini coefficient     .7985693    .7131869
                  MSE     .1126847    .1314475
                 RMSE     .3356854    .3625569
```

By tuning, we increased the cross-validation AUCPR from 0.658 to 0.674. The improvement is small, because we explored only a small portion of the hyperparameter space in this example. Hyperparameter tuning is an iterative process that requires many iterations to sufficiently explore the hyperparameter space.

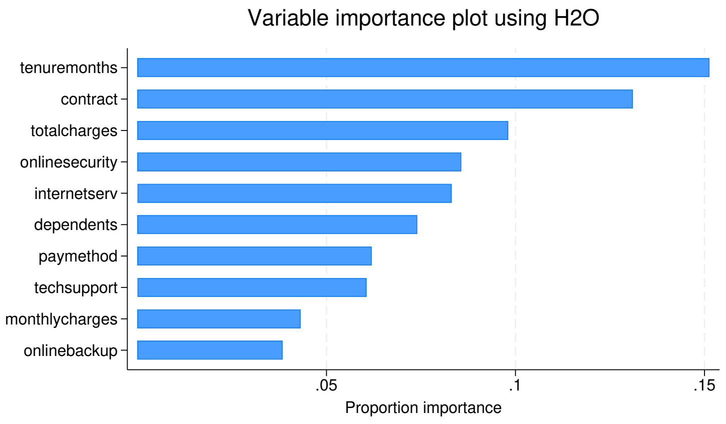
Let’s compare the best model, which we store as gbm_tuned, with the reference model from the previous example based on other metrics by using the h2omlgof command.

```
. h2omlest store gbm_tuned
. h2omlgof gbm_default gbm_tuned
Performance metrics for model comparison using H2O
Training frame: train
───────────────────────────────────────────────
                      │  gbm_def~t   gbm_tuned
──────────────────────┼────────────────────────
Training              │
  No. of observations │      5,643       5,643
             Log loss │   .3293387    .3531063
     Mean class error │   .1603572    .1784776
                  AUC │   .9163226    .8992847
                AUCPR │   .8023966    .7610732
     Gini coefficient │   .8326452    .7985693
                  MSE │   .1034999    .1126847
                 RMSE │    .321714    .3356854
──────────────────────┼────────────────────────
Cross-validation      │
  No. of observations │      5,643       5,643
             Log loss │    .411338    .4026141
     Mean class error │   .2338787    .2313897
                  AUC │   .8500772    .8565935
                AUCPR │   .6584908     .673929
     Gini coefficient │   .7001545    .7131869
                  MSE │   .1350446    .1314475
                 RMSE │   .3674841    .3625569
───────────────────────────────────────────────
```

In the output, the first section reports the training results, and the second section reports the cross-validation results. Looking at the cross-validation results, we see that tuning improved the model performance for all metrics. The log loss, mean of per-class error rates, mean squared error (MSE), and root mean squared error (RMSE) are all smaller for the tuned model, whereas area under the curve (AUC), AUCPR, and the Gini coefficient are larger for the tuned model, all of which indicate better performance.

In addition to tuning, we may also refine the list of predictors based on variable importance.

```
. h2omlgraph varimp
```



Based on the above graph, we may decide to drop the predictor onlinebackup.

Variable selection with cross-validation requires careful implementation to avoid so-called data leakage, where the training data contain information that would not be available during prediction on the testing data; see Raschka (2020) for details.

◁

▷ Example 4: Method selection and prediction

In example 5 of [H2OML] ***h2oml gbm***, we used hyperparameter tuning to select the best GBM model. Instead of GBM, we may consider other methods such as random forest or logistic regression. In this example, we compare GBM and random forest.

Instead of tuning the random forest model following similar steps from example 5 of [H2OML] ***h2oml gbm***, for simplicity, we pretend that the following model is our tuned model for random forest. We also store the working model as `rf_tuned` by using the `_h2omlest store` command.

```
. h2oml rfbinclass churn $predictors, h2orseed(19) cv(3, stratify)
> ntrees(200) minobsleaf(2)

Progress (%): 0 7.1 14.1 19.8 24.8 56.2 75.0 79.8 84.7 89.4 93.9 100

Random forest binary classification using H2O

Response: churn
Frame:                                    Number of observations:
  Training: train                                    Training =  5,643
                                             Cross-validation =  5,643
Cross-validation: Stratify                Number of folds     =      3

Model parameters

Number of trees       =  200
             actual   =  200
Tree depth:                               Pred. sampling value =    -1
           Input max  =   20              Sampling rate        =   .632
                 min  =   16              No. of bins cat.     =  1,024
                 avg  = 19.6              No. of bins root     =  1,024
                 max  =   20              No. of bins cont.    =     20
Min. obs. leaf split  =    2              Min. split thresh.   = .00001

Metric summary

                                            Cross-
              Metric     Training       validation

            Log loss     .4153088         .416142
    Mean class error     .2396365         .230295
                 AUC     .8507327        .8453018
               AUCPR     .6526923        .6452846
    Gini coefficient     .7014654        .6906036
                 MSE     .1335578        .1358418
                RMSE     .3654556        .3685673

. h2omlest store rf_tuned
```

To choose the best method, we compute performance metrics using the testing frame. To compute AUCPR for the testing frame, we use the `h2omlpostestframe` command to specify the name of the frame, `test` in our case, to be used by a subset of postestimation commands for computations.

```
. h2omlpostestframe test
(testing frame test is now active for h2oml postestimation)
```

By default, the specified frame is considered to be a testing frame and is labeled as “Testing” in the output, but you can specify your own label by using the `framelabel()` option. To report the metrics for the selected testing frame, we use the `h2omlestat metrics` command.

```
. h2omlestat metrics

Performance metrics using H2O
Random forest binary classification

Response:      churn
Testing frame: test

Number of observations = 1,400

─────────────────────────────────
               Metric │   Testing
──────────────────────┼──────────
             Log loss │  .4101135
     Mean class error │  .2241742
                  AUC │    .85292
                AUCPR │  .6847162
     Gini coefficient │    .70584
                  MSE │  .1328891
                 RMSE │  .3645396
─────────────────────────────────
```

We next compute the metrics for the testing frame for the GBM model after restoring its estimation results.

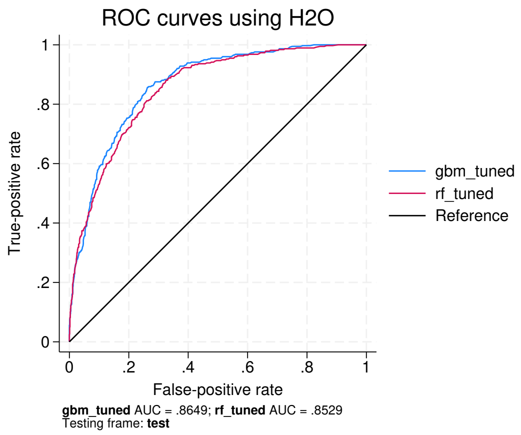
```
. h2omlest restore gbm_tuned
(results gbm_tuned are active now)

. h2omlpostestframe test
(testing frame test is now active for h2oml postestimation)

. h2omlestat metrics

Performance metrics using H2O
Gradient boosting binary classification

Response:      churn
Loss:          Bernoulli
Testing frame: test

Number of observations = 1,400

─────────────────────────────────
               Metric │   Testing
──────────────────────┼──────────
             Log loss │  .3964014
     Mean class error │  .2030941
                  AUC │  .8649185
                AUCPR │  .6963289
     Gini coefficient │  .7298371
                  MSE │  .1284349
                 RMSE │  .3583782
─────────────────────────────────
```

We can compare the results side by side more easily by using the h2omlgof command.

```
. h2omlgof rf_tuned gbm_tuned

Performance metrics for model comparison using H2O
Testing frame: test

──────────────────────────────────────────────
                      │   rf_tuned  gbm_tuned
──────────────────────┼───────────────────────
Testing               │
  No. of observations │      1,400      1,400
             Log loss │   .4101135   .3964014
     Mean class error │   .2241742   .2030941
                  AUC │     .85292   .8649185
                AUCPR │   .6847162   .6963289
     Gini coefficient │     .70584   .7298371
                  MSE │   .1328891   .1284349
                 RMSE │   .3645396   .3583782
──────────────────────────────────────────────
```

Based on this example, GBM outperforms random forest because AUCPR for GBM is higher. Thus, we choose GBM as our selected best method. We can also compare methods (or models) based on ROC curves, which plots the true-positive rate versus false-positive rate for different thresholds. The closer the curve to the upper left corner, the better the model fit. Because the test frame has been set for both models, the reported results correspond to the testing frame. For details, see [H2OML] **h2omlgraph roc**.

```
. h2omlgraph roc, models(gbm_tuned rf_tuned)
```



Based on the ROC results, as we expected, the GBM method slightly outperforms the random forest method.

Another popular approach to compare classification predictions between different methods and models is by using a confusion matrix, which reports the numbers of correctly and incorrectly predicted outcomes. Below, we use `h2omlestat confmatrix` to produce the confusion matrix after the GBM estimation for the testing frame we selected earlier with `h2omlpostestframe`.

```
. h2omlestat confmatrix

Confusion matrix using H2O
Testing frame: test

             |      Predicted
       churn |         No          Yes |   Total   Error     Rate
-------------+-------------------------+---------------------------
          No |        754          269 |   1,023     269     .263
         Yes |         54          323 |     377      54     .143
-------------+-------------------------+---------------------------
       Total |        808          592 |   1,400     323     .231
Note: Probability threshold .2378 that maximizes F1
      metric used for classification.
```

In H2O, the “positive” class corresponds to the second label in lexicographical order, which in our case is `Yes`. To see the levels of the categorical variable, type

```
. _h2oframe levelsof churn
`"No"' `"Yes"'
```

From the output, 323 and 754 correspond to true-positive and true-negative responses, respectively, and the misclassification error rate is 0.231. By default, the threshold for binary classification of 0.2378 is selected based on maximizing the F1 metric. Observations with predicted values above this threshold will be classified as “Yes”, and the remaining observations will be classified as “No”. You may want to see the results based on a different metric. For instance, consider a scenario where a company uses predictions to offer additional discounts or free services to customers who are likely to churn. If these benefits are costly, the company would prioritize predictions that maximize precision. To report the confusion matrix using a different metric, use the `metric()` option.

We encourage you to perform the same analysis for the `rf_tuned` model to verify that GBM indeed outperforms random forest on the testing frame.

◁

▷ Example 5: Classification prediction on new data

Continuing with example 4, suppose the company collected new data stored in `newchurn.dta`. It wants to predict the probability of churn for these new customers based on the GBM model `gbm_tuned`.

Let's read the new dataset as an H2O frame and list the first two observations to see some of the new data by using the `_h2oframe list` command.

```
. use https://www.stata-press.com/data/r19/newchurn
(Telco customer churn new data)

. _h2oframe put, into(newchurn) replace

Progress (%): 0 100

. _h2oframe change newchurn

. _h2oframe list in 1/2

  zipcode    latitude     longitude  tenure~s  monthlyc~s  totalcharges
1   95670  38.6027222  -121.2799149        49  75.1999969  3678.3000488
2   91737  34.2452888  -117.6425018         4  88.8499985   372.4500122

        country       state                city  gender  senior~n  partner
1 United States  California    Rancho Cordova    Male        No       No
2 United States  California  Rancho Cucamonga  Female       Yes       No

  depend~s  phones~e  multip~s  internets~v  online~y  online~p  device~t
1       No       Yes       Yes  Fiber optic        No        No        No
2       No       Yes       Yes  Fiber optic        No        No       Yes

  techsu~t  streamtv  stream~e        contract  paperl~l         paymethod
1       No        No        No  Month to month        No       Credit card
2       No        No       Yes  Month to month       Yes  Electronic check

[2 rows x 25 columns]
```

The probabilities of churning and the corresponding classes can be predicted by using the `h2omlpredict` command. By default, this command predicts classes after classification. To predict probabilities instead, we need to specify the `pr` option with `h2omlpredict`. In example 4, we used `h2omlpostestframe` to set the postestimation frame to test for the `gbm_tuned` model. To obtain predictions for the new dataset, specify the `frame(newchurn)` option with `h2omlpredict`. Below, we predict both classes and probabilities for the new dataset using the `gbm_tuned` model.

```
. h2omlest restore gbm_tuned
(results gbm_tuned are active now)

. h2omlpredict churnhat, frame(newchurn)
(option class assumed; predicted class)

Progress (%): 0 100

. h2omlpredict churnprob*, frame(newchurn) pr

Progress (%): 0 100
```

By default, the threshold that maximizes the F1 metric is used to predict classes based on the predicted probabilities. You can specify a different value for the threshold using the threshold() option. To display the threshold values that maximize or minimize different classification metrics, we type

```
. h2omlestat threshmetric

Maximum or minimum metrics using H2O
Testing frame: test

─────────────────────────────┬──────────────────────
                      Metric │   Max/Min   Threshold
─────────────────────────────┼──────────────────────
                          F1 │     .6667       .2378
                          F2 │     .7816       .1496
                        F0.5 │     .6659       .5142
                    Accuracy │     .8171       .5142
                   Precision │         1       .9081
                      Recall │         1       .0236
                 Specificity │         1       .9081
         Min. class accuracy │     .7849       .2905
         Mean class accuracy │     .7969       .2378
              True negatives │      1023       .9081
             False negatives │         0       .0236 +
              True positives │       377       .0236
             False positives │         0       .9081 +
          True-negative rate │         1       .9081
         False-negative rate │         0       .0236 +
          True-positive rate │         1       .0236
         False-positive rate │         0       .9081 +
                         MCC │     .5332       .2378
─────────────────────────────┴──────────────────────
+ identifies minimum metrics.
```

The table above displays the set of classification metrics with the corresponding best thresholds; see [H2OML] **h2omlestat threshmetric**. In the reported table, the thresholds provide the best cutpoints for the classification based on the predicted probabilities such that the corresponding metric is optimal. For example, for Precision, the best threshold is 0.9081. For the definition of metrics, see [H2OML] ***metric_option***.

The generated variables for the classes and class probabilities are available in the newchurn frame, because we specified frame(newchurn). Let's list a few values for the predicted classes and probabilities.

```
. _h2oframe list churnhat churnprob*

  churnhat  churnp~1  churnp~2
1       No  .7780746  .2219254
2      Yes  .2161581  .7838419
3       No  .9001728  .0998272
4       No  .8937768  .1062232
5       No  .8101463  .1898537
6      Yes  .2203342  .7796658
7       No  .8987335  .1012665
8      Yes  .4977883  .5022117

[8 rows x 3 columns]
```

The variables (H2O columns) churnhat, churnprob1, and churnprob2 contain the predicted classes and the corresponding predicted probabilities of not churning or churning. In our example, for instance, there is only a 22% chance that the first customer will churn compared with a 78% chance of churning for the second customer.

◁

▷ Example 6: Explaining classification prediction

In this example, we try to answer one of the fundamental questions of machine learning: Why does my model predict what it predicts? In machine learning, explainability refers to the ability of the method to describe how a model arrives at a specific prediction in a way that is understandable to humans. This is important to ensure that, under certain conditions, predictions are not only accurate but also understandable and justifiable.

From *Interpretation and explanation* in [H2OML] **Intro**, there are two types of explainability methods: local and global. Local models explain individual predictions and approximate the machine learning model in the vicinity of one observation. The popular methods include ICE curves and SHAP values, which can be obtained by using the h2omlgraph ice and h2omlgraph shapvalues commands. A global model describes an average behavior of a machine learning model. PDPs, variable importance, and global surrogate models are some of the popular choices.

We start with global methods and then switch to local methods. In example 4, we selected gbm_tuned as the best model. In this example, we want to explore predictions for the original churn dataset (without splitting it into training and testing frames). We start by restoring the gbm_tuned model:

```
. h2omlest restore gbm_tuned
(results gbm_tuned are active now)
```

Now we use h2omlpredict to predict classes for the entire churn dataset. We specify the frame() option to obtain predictions for the churn frame instead of the test frame we selected with h2omlpostestframe earlier in example 4.

```
. h2omlpredict churnhat, frame(churn)
(option class assumed; predicted class)
```

We use these predictions to build global surrogate models, which are some of the simplest global explainable methods. They approximate the prediction of a machine learning model, churnhat in our case, using a model that is easier to interpret such as a decision tree. See *Global surrogate models* in [H2OML] **Intro**.

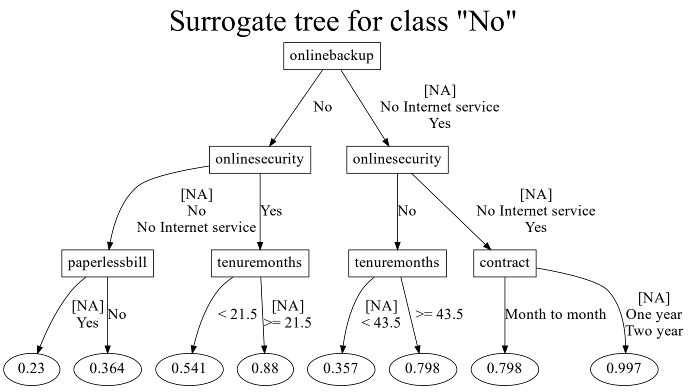
To demonstrate, we use a classification tree with maximum depth equal to, say, 3 and other parameters at their default values as a global surrogate model. In practice, the depth of the tree and other parameters should be treated as hyperparameters and learned from data. To obtain one classification tree, we use the ntrees(1) option with h2oml rfbinclass.

In example 1, we set our working frame as `train`. Thus, before running the estimation command `h2oml rfbinclass` on the `churn` dataset, we need to physically change the working frame to `churn` by using the `_h2oframe change` command.

```
. _h2oframe change churn

. h2oml rfbinclass churnhat $predictors, h2orseed(19) ntrees(1) maxdepth(3)

Progress (%): 0 100

Random forest binary classification using H2O

Response: churnhat
Frame:                                      Number of observations:
  Training: churn                                      Training =  2,523

Model parameters

Number of trees      =   1
              actual =   1
Tree depth:                                 Pred. sampling value =     -1
           Input max =   3                  Sampling rate        =   .632
                 min =   3                  No. of bins cat.     =  1,024
                 avg = 3.0                  No. of bins root     =  1,024
                 max =   3                  No. of bins cont.    =     20
Min. obs. leaf split =   1                  Min. split thresh.   = .00001

Metric summary

------------------------------
                 |
          Metric |   Training
-----------------+------------
                 |
        Log loss |   .4182261
Mean class error |   .1828537
             AUC |   .8678704
           AUCPR |    .727738
Gini coefficient |   .7357409
             MSE |   .1378874
            RMSE |   .3713319
------------------------------
```

It is easier to interpret the results from a classification tree visually. The steps on how to obtain an image from the DOT file are provided in [H2OML] **DOT extension**. We follow those steps to display the classification tree below; see [H2OML] **h2omltree**. The `dotsaving()` option of the `h2omltree` command generates and saves a DOT file, which can be used to plot the classification tree using the Graphviz software, see https://graphviz.org.

```
. h2omltree, dotsaving(churntree.dot, replace
> title(Surrogate tree for class "No"))
```

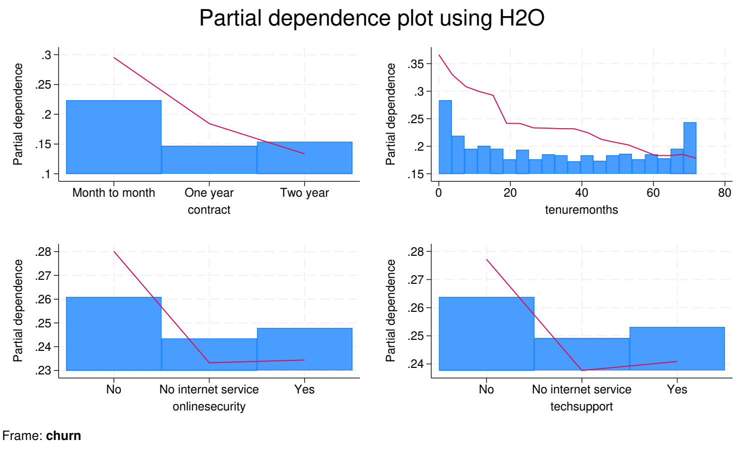


The NA's on the tree indicate the split for the missing values, if any. The values of the terminal nodes can be interpreted as probabilities of class `No`. For example, the highest-predicted probability of not churning (0.997) or the lowest probability of churning ($1 - 0.997 = 0.003$) occurs for the customers who have a one- or two-year contract with the company and are either not subscribed to any internet services or use online backup and online security services.

In example 3, we used `h2omlgraph varimp` to display important predictors for the `gbm_tuned` model. We use some of these important predictors to produce PDP. PDP is a global explainable method that shows the marginal effect that the specified predictors have on the predicted outcome of a machine learning model (`gbm_tuned` here); see [H2OML] **h2omlgraph pdp**.

Our current estimation results are from the h2oml rfbinclass command, so we first use h2omlest restore to restore the gbm_tuned estimation results. Next we use h2omlpostestframe with the notest option to specify that the churn frame be used by the subsequent postestimation commands but not considered a testing frame.

```
. h2omlest restore gbm_tuned
(results gbm_tuned are active now)
. h2omlpostestframe churn, notest
(frame churn is now active for h2oml postestimation)
. h2omlgraph pdp contract tenuremonths onlinesecurity techsupport, combine
Progress (%): 0 75.0 100
```



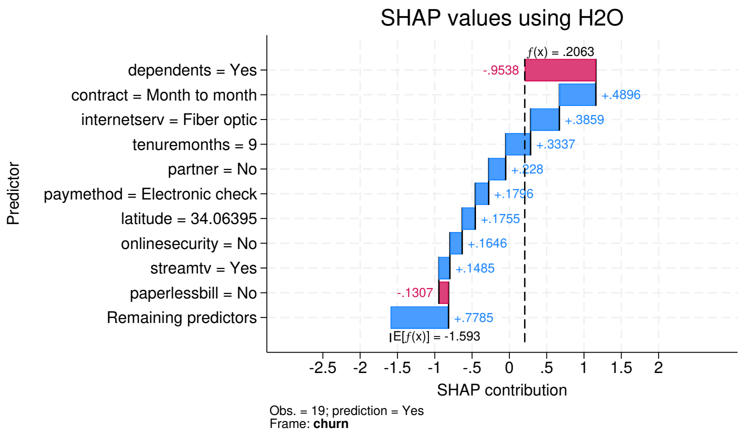
The PDP pattern (red line in the plot) agrees with the results from the surrogate tree. For instance, the probability of churning (shown on the y axis) decreases for customers with a one- or two-year contract (contract) and for customers who use the company's services longer (tenuremonths).

For local explainability, we can use SHAP values. A SHAP value estimates the contribution of each predictor to the prediction for an individual observation. Let's consider observation 19 and explain its prediction from the gbm_tuned model. Below, we list some of the predictors for this observation, which corresponds to a female customer who used a month-to-month contract service for 9 months and has both the observed churn and predicted churnhat values of Yes.

```
. _h2oframe list churn churnhat contract totalcharges onlinesecurity
> tenuremonths gender in 19
  churn  churnhat        contract  totalc~s  online~y  tenure~s  gender
1   Yes       Yes  Month to month    857.25        No         9  Female
[1 row x 7 columns]
```

We now use `h2omlgraph shapvalues` to produce SHAP values for observation 19 for the top 10 SHAP-important predictors.

```
. h2omlgraph shapvalues, obs(19) top(10) xlabel(-2.5(0.5)2)
```

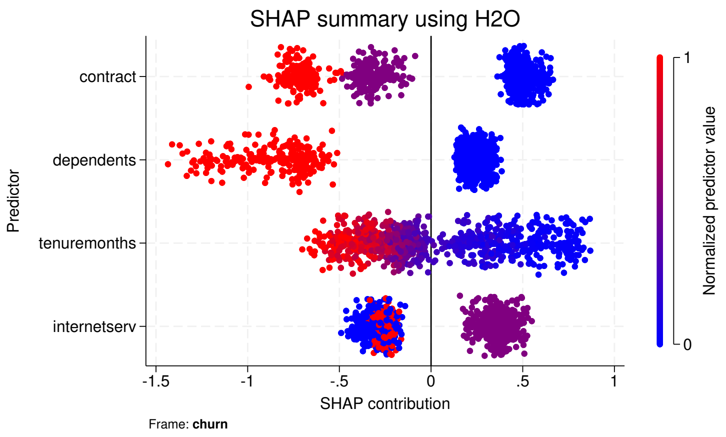


The blue bars show predictors that increase probability of churn, and red bars indicate the opposite. The SHAP values agree with previous findings. Month-to-month `contract`, small `tenuremonths`, and not using online security services contribute positively to this particular customers' churning. On the other hand, having a dependent contributes to retaining this particular customer to continue using the company's services.

We can also display the SHAP summary plot, also known as a beeswarm plot, for all observations and predictors. The beeswarm plot shows both the magnitudes of SHAP values, which represent the contribution of a predictor to a particular prediction, and the SHAP-value distribution across many observations. This allows you to quickly see which predictors are most important and how they influence the response.

For illustration purposes, we plot SHAP values for the top 4 SHAP-important predictors.

```
. h2omlgraph shapsummary, top(4) rseed(19)
```



In the figure, the color map, titled as “Normalized predictor value”, indicates colors of the normalized values of the predictors. For example, if a variable is not of the data type `enum`, such as `tenuremonths`, then the smallest normalized variable value will be given a lighter blue color, and, as the values increase, the color gradient will change from blue to red for the largest value of 1. Similarly, for a categorical variable (`enum`), such as `contract`, the base level of the predictor will be given a lighter blue color, and the color will change from blue to red according to the categories. Within each level, the observations are jittered for presentational purposes. To check the levels of a categorical variable (for example, `contract`), type

```
. _h2oframe levelsof contract
`"Month to month"' `"One year"' `"Two year"'
```

The predictors displayed on the y axis are ranked based on SHAP predictor importance: predictors with large absolute SHAP values are listed in descending order. From the SHAP summary plot, for the `contract` predictor, a smaller value, which corresponds to the month-to-month option, increases the probability of churn, and this probability decreases for the other contract options. Similarly, smaller values of `tenuremonths` increase the probability of churn and vice versa.

◁

▷ Example 7: Shutting down the H2O cluster

Once you are finished with your analysis, you can disconnect from the H2O cluster by using

```
. h2o disconnect
```

This command closes the H2O session between Stata and the cluster. However, the H2O cluster continues running in the background. Later in the same Stata session, you can type `h2o connect` to rebuild the connection to it and reaccess the resources it contains. If you want to force shutting down the cluster, you can type

```
. h2o shutdown, force
```

The above completely shuts down the cluster, and all resources within the cluster are lost, including any data (H2O frames) it contained.

If you want the H2O cluster to remain connected but would like to clear everything in memory, including all data in H2O frames, you can type

```
. h2o clear
```

◁

Regression analysis

In this section, we demonstrate analysis for the regression problem using random forest.

▷ Example 8: Data setup

Consider the Ames housing dataset (De Cock 2011), `ameshouses.dta`, also used in a Kaggle competition, which describes residential houses sold in Ames, Iowa, between 2006 and 2010. It contains about 80 housing (and related) characteristics such as home size, amenities, and location. This dataset is often used for building predictive models for home sale price, `saleprice`. We will use random forest to model home sale price and evaluate its predictive performance. Here we will use just a few predictors to demonstrate some of the `h2oml` features.

Before putting the dataset into an H2O frame, we do several data transformations in Stata. In particular, because `saleprice` is right-skewed (type `histogram saleprice`), we perform logarithmic transformation. We also generate the `houseage` variable, which records the age of the house at the time of a sales transaction.

```
. use https://www.stata-press.com/data/r19/ameshouses
(Ames house data)
. generate logsaleprice = log(saleprice)
. generate houseage = yrsold - yearbuilt
. drop saleprice yearbuilt yrsold
```

We put the dataset into an H2O frame by using the `_h2oframe put` command. We split the data into training and validation frames (without a testing frame) with 75% of observations in the training frame.

```
. h2o init
  (output omitted)
. _h2oframe put, into(house)
Progress (%): 0 100
. _h2oframe change house
. _h2oframe split house, into(train valid) split(0.75 0.25) rseed(19)
. _h2oframe change train
```

The steps of method selection and prediction for the regression are the same as for binary classification, discussed in example 3 and example 4. Therefore, in this example, we focus only on tuning.

◁

▷ Example 9: Regression using random forest

As we discussed in *Model selection in machine learning* of [H2OML] **Intro**, we start by defining a reference model, which in our case is a random forest with default parameters. We use the MSE metric, computed on validation frame, to evaluate the performance of the model.

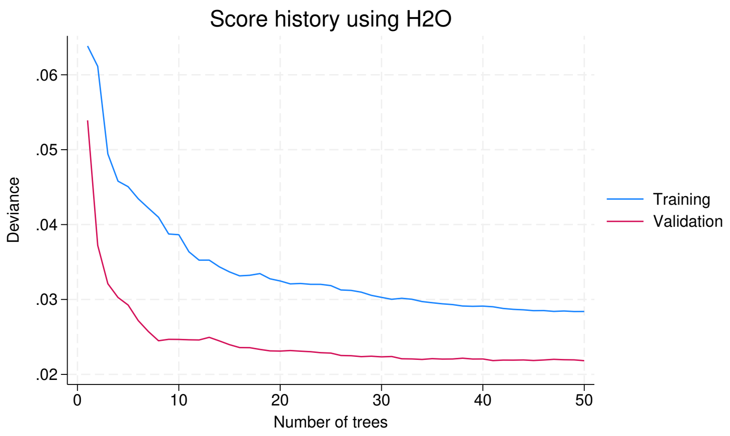
The dataset has a total of 46 predictors, but for simplicity, we include only 10 and create a global macro, `predictors`, in Stata to store the names of these predictors.

```
. global predictors overallqual  grlivarea exterqual houseage garagecars
> totalbsmtsf stflrsf garagearea kitchenqual bsmtqual

. h2oml rfregress logsaleprice $predictors, h2orseed(19) validframe(valid)

Progress (%): 0 54.0 100

Random forest regression using H2O

Response: logsaleprice
Frame:                                 Number of observations:
  Training:   train                               Training =  1,099
  Validation: valid                             Validation =    361

Model parameters

Number of trees      =   50
              actual =   50
Tree depth:                            Pred. sampling value =     -1
           Input max =   20            Sampling rate        =   .632
                 min =   18            No. of bins cat.     =  1,024
                 avg = 19.9            No. of bins root     =  1,024
                 max =   20            No. of bins cont.    =     20
Min. obs. leaf split =    1            Min. split thresh.   = .00001

Metric summary

-----------------------------------
    Metric |   Training  Validation
-----------+-----------------------
  Deviance |   .0283991    .0218303
       MSE |   .0283991    .0218303
      RMSE |   .1685202    .1477508
     RMSLE |   .0130751    .0114914
       MAE |   .1163998    .1042066
 R-squared |   .8240197    .8577693
-----------------------------------
```

The description and interpretation of the output of random forest is provided in example 1 of [H2OML] ***h2oml rf***. The definitions of metrics can be found in [H2OML] ***metric_option***.

The MSE for the validation frame is 0.022, which is our reference value for later. We also need to make sure that we are slightly overfitting the training dataset. The above model does not overfit the training dataset, because the training MSE is larger than the validation MSE. To visualize this, we plot the validation curve using the `h2omlgraph scorehistory` command.

```
. h2omlgraph scorehistory
Training frame:   train
Validation frame: valid
```



We observe that the training error is higher than the validation error. This means that either the default model is not complex enough to overfit the training dataset or we need more training data. In our case, the former reason is more likely, because we used a simpler model with default hyperparameters, which is sufficient for a reference model.

◁

▷ Example 10: Hyperparameter tuning using random forest

In this example, we explore different configurations of the hyperparameters to tune the random forest model. In general, a well-tuned model substantially improves the model performance and generalizes well to new data.

To demonstrate, we tune only two hyperparameters, the number of trees, `ntrees()`, and the minimum number of observations required for splitting a leaf node, `minobsleaf()`, and use a small grid space with a random grid search. In practice, hyperparameter tuning is an iterative process and often requires tuning many more hyperparameters; see table 3 in [H2OML] **Intro**. When the number of hyperparameters and the grid space are large, you can use the `parallel()` option to specify the number of models to build in parallel during the grid search. Beware that the H2O results for models built in parallel may not always be reproducible; see [H2OML] **H2O reproducibility**. By default, the models are built sequentially, which may take some time for complicated tuning models.

```
. h2oml rfregress logsaleprice $predictors, h2orseed(19) validframe(valid)
> ntrees(400(50)500) minobsleaf(3(2)7)
> tune(grid(random, h2orseed(19)) metric(mse))
Progress (%): 0 100

Random forest regression using H2O

Response: logsaleprice
Frame:                                  Number of observations:
  Training:   train                                Training =  1,099
  Validation: valid                              Validation =    361

Tuning information for hyperparameters

Method: Random
Metric: MSE

─────────────────────────────────────────────────────────────────────
                      │                 Grid values
      Hyperparameters │        Minimum        Maximum        Selected
──────────────────────┼──────────────────────────────────────────────
      Number of trees │            400            500             450
  Min. obs. leaf split│              3              7               3
─────────────────────────────────────────────────────────────────────

Model parameters

Number of trees      =  450
          actual     =  450
Tree depth:                             Pred. sampling value =     -1
           Input max =   20             Sampling rate        =   .632
                 min =   12             No. of bins cat.     =  1,024
                 avg = 15.1             No. of bins root     =  1,024
                 max =   20             No. of bins cont.    =     20
Min. obs. leaf split =    3             Min. split thresh.   = .00001

Metric summary
───────────────────────────────────
     Metric │   Training  Validation
────────────┼──────────────────────
   Deviance │   .0269402    .0208756
        MSE │   .0269402    .0208756
       RMSE │   .1641346     .144484
      RMSLE │   .0127415    .0112297
        MAE │   .1113531    .0995714
  R-squared │     .83306    .8639893
───────────────────────────────────
```

To ensure H2O reproducibility, we specified `h2orseed(19)` for both the random forest model and grid search. Despite tuning only a couple hyperparameters, we were able to reduce the validation MSE metric from 0.022 to 0.021. To explore tuning further, you may try to include more hyperparameters and consider a larger grid space.

To compare different configurations of hyperparameters with their respective metric values sorted from the most to least optimal, we can use the h2omlestat gridsummary command.

```
. h2omlestat gridsummary
Grid summary using H2O
────────────────────────────────────
                        Min. obs.
        Number of            leaf
 ID         trees           split          MSE
────────────────────────────────────
  1           450               3     .0208756
  2           500               3     .0209012
  3           400               3      .020924
  4           400               5      .021525
  5           450               5     .0215336
  6           500               5     .0215765
  7           500               7     .0221419
  8           400               7      .022142
  9           450               7     .0221425
────────────────────────────────────
```

Here the hyperparameter values are listed from the smallest to largest MSE. If you want to reduce execution time in favor of a slightly lower model performance, you may select the third model instead of the first (top) model. For this model, the number of trees is 400 compared with 450 for the top model, but the MSE value is only slightly higher. We can select the third model for further analysis by typing

```
. h2omlselect id = 3
```

◁

Effect of categorical predictors

As we discussed in *Decision trees* of [H2OML] **Intro**, the ensemble decision tree methods are biased toward categorical predictors with many levels. In this example, we explore the effect of a categorical predictor with many levels on performance of tree-based methods. Even though we focus on a GBM here, similar results should also hold for a random forest.

▷ Example 11: Data setup

We use a subset of the Lending Club dataset available in Kaggle to explore this phenomenon. Kaggle is a platform for the machine learning community that provides datasets and other resources; see https://kaggle.com.

We start by initializing an H2O cluster and importing the dataset as an H2O frame by using the h2o init and _h2oframe put commands.

```
. h2o init
. use https://www.stata-press.com/data/r19/loan
(Lending club data)
. _h2oframe put, into(loan)
Progress (%): 0 100
```

Next we use the _h2oframe split command to split the dataset into training and validation frames with 80% of observations in the training frame.

```
. _h2oframe split loan, into(train valid) split(0.8 0.2) rseed(19)
. _h2oframe change train
```

◁

▷ Example 12: Effect of categorical predictors on ensemble decision tree methods

Consider the categorical predictor addr_state with 50 levels that records the state where the loan applicant lives. To show the importance of carefully treating categorical variables when performing ensemble decision tree methods, we first run a GBM without paying special attention to categorical predictors.

Let's define a global macro, predictors, to store the names of the predictors.

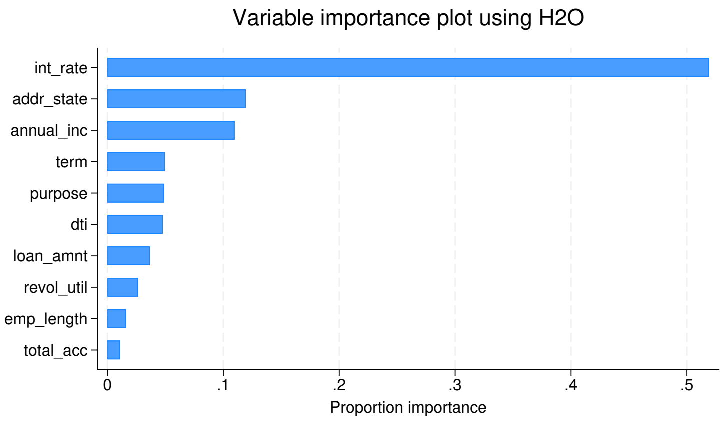
```
. global predictors loan_amnt int_rate emp_length annual_inc dti delinq_2yrs
> revol_util total_acc credit_lngth term home_owner purpose addr_state
> verification
```

Next we use h2oml gbbinclass to perform gradient boosting binary classification. We perform validation using the valid frame and specify the h2orseed() option for H2O reproducibility. We use 200 trees, and, to avoid overfitting, we request an early stopping based on the AUC metric. We also specify scoreevery(1) to score the AUC metric after each tree is added to the model to ensure H2O reproducibility in the presence of early stopping.

```
. h2oml gbbinclass bad_loan $predictors, h2orseed(19) validframe(valid)
> ntrees(200) stop(5, metric(auc)) scoreevery(1)
Progress (%): 0 1.4 5.0 10.9 18.0 100

Gradient boosting binary classification using H2O

Response: bad_loan
Loss:     Bernoulli
Frame:                                       Number of observations:
  Training:   train                                     Training = 131,294
  Validation: valid                                   Validation =  32,693

Model parameters

Number of trees      = 200                   Learning rate       =      .1
              actual =  39                   Learning rate decay =       1
Tree depth:                                  Pred. sampling rate =       1
           Input max =   5                   Sampling rate       =       1
                 min =   5                   No. of bins cat.    =   1,024
                 avg = 5.0                   No. of bins root    =   1,024
                 max =   5                   No. of bins cont.   =      20
Min. obs. leaf split =  10                   Min. split thresh.  =  .00001

Stopping criteria:                           No. of iterations   =       5
  Metric: AUC                                Tolerance           =    .001

Metric summary

           Metric   Training  Validation

         Log loss   .4256225    .4381805
 Mean class error   .3405512    .3471389
              AUC   .7264524    .7081155
            AUCPR   .3827862    .3495525
 Gini coefficient   .4529049    .4162309
              MSE   .1337261    .1384392
             RMSE   .3656858    .3720742

Note: Metric is scored after every tree.
```

Let's plot the variable importance by using the `h2omlgraph varimp` command.

```
. h2omlgraph varimp
```



The variable `addr_state` is one of the important variables.

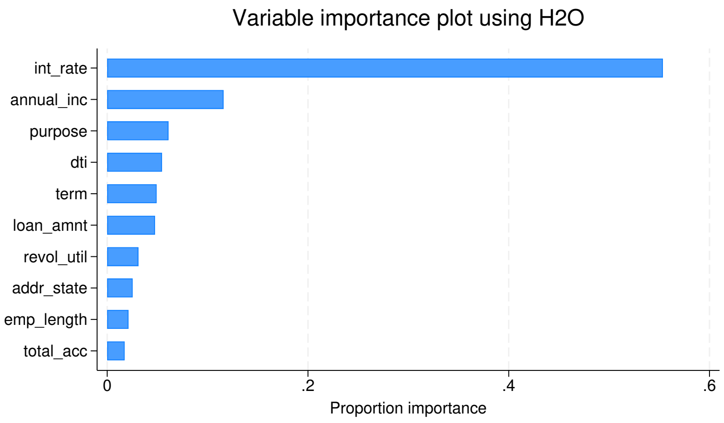
Now to account for the many categories in `addr_state`, we tune the hyperparameter `binscat()` on a grid of values $[16, 50]$.

```
. h2oml gbbinclass bad_loan $predictors, h2orseed(19) validframe(valid)
> ntrees(200) binscat(16(5)50) stop(5, metric(auc)) scoreevery(1)
> tune(grid(cartesian) metric(auc))
Progress (%): 0 100

Gradient boosting binary classification using H2O

Response: bad_loan
Loss:     Bernoulli
Frame:                                       Number of observations:
  Training:   train                                     Training = 131,294
  Validation: valid                                   Validation =  32,693

Tuning information for hyperparameters

Method: Cartesian
Metric: AUC

───────────────────┬──────────────────────────────────────────────────
                   │                     Grid values
   Hyperparameters │         Minimum           Maximum         Selected
───────────────────┼──────────────────────────────────────────────────
  No. of bins cat. │              16                46               46
───────────────────┴──────────────────────────────────────────────────

Model parameters

Number of trees       = 200              Learning rate       =       .1
              actual =   46              Learning rate decay =        1
Tree depth:                              Pred. sampling rate =        1
           Input max =    5              Sampling rate       =        1
                 min =    5              No. of bins cat.    =       46
                 avg =  5.0              No. of bins root    =    1,024
                 max =    5              No. of bins cont.   =       20
Min. obs. leaf split =   10              Min. split thresh.  =   .00001

Stopping criteria:                       No. of iterations   =        5
  Metric: AUC                            Tolerance           =     .001

Metric summary

───────────────────┬──────────────────────
            Metric │  Training  Validation
───────────────────┼──────────────────────
          Log loss │  .4274797    .4368557
  Mean class error │  .3422759    .3435895
               AUC │  .7210886    .7100941
             AUCPR │  .3725785    .3557051
  Gini coefficient │  .4421772    .4201882
               MSE │  .1344013    .1379741
              RMSE │  .3666078    .3714487
───────────────────┴──────────────────────
Note: Metric is scored after every tree.
```

Based on the tuning information, the value of 46 for `binscat()` provides the highest AUC value.

The variable importance graph for the selected best model, displayed below, shows that after accounting for the many levels of the categorical variable addr_state, its importance has decreased substantially.

```
. h2omlgraph varimp
```



◁

Detecting nuisance predictors

▷ Example 13: Detecting nuisance predictors with ensemble decision tree methods

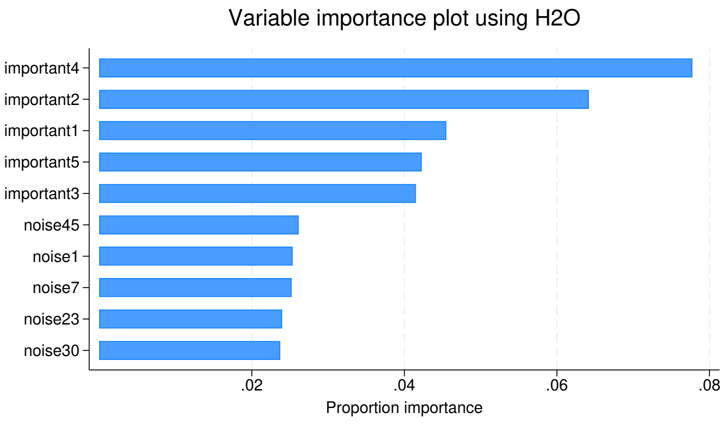
Let's use ensemble decision trees to detect important and nuisance predictors in the dataset. Here we use a random forest, but the results should be similar for a GBM as well. We use a simulated dataset, in which predictors important1 through important5 are important and noise1 through noise5 are nuisance (random noise). For the data-generation details, see Wright, Ziegler, and König (2016).

We start by initializing an H2O cluster and importing the dataset as an h2oframe.

```
. use https://www.stata-press.com/data/r19/effect
(Simulated data with many nuisance predictors)
. h2o init
  (output omitted)
. _h2oframe put, into(sim)
Progress (%): 0 100
. _h2oframe change sim
```

Next we run a random forest binary classification with default hyperparameter values and plot the variable importance.

```
. h2oml rfbinclass y important1-important5 noise1-noise45, h2orseed(19)
Progress (%): 0 47.9 100
Random forest binary classification using H2O
Response: y
Frame:                                  Number of observations:
  Training: sim                                     Training =  1,000
Model parameters
Number of trees       =   50
             actual =   50
Tree depth:                             Pred. sampling value =     -1
           Input max =   20             Sampling rate        =   .632
                 min =   15             No. of bins cat.     =  1,024
                 avg = 18.6             No. of bins root     =  1,024
                 max =   20             No. of bins cont.    =     20
Min. obs. leaf split =    1             Min. split thresh.   = .00001
Metric summary
─────────────────────────────
              Metric │ Training
─────────────────────┼───────
            Log loss │ .6693054
    Mean class error │ .3711672
                 AUC │  .689691
              AUCPR │ .6739805
    Gini coefficient │ .3793821
                 MSE │ .2227112
                RMSE │ .4719228
─────────────────────────────

. h2omlgraph varimp
```

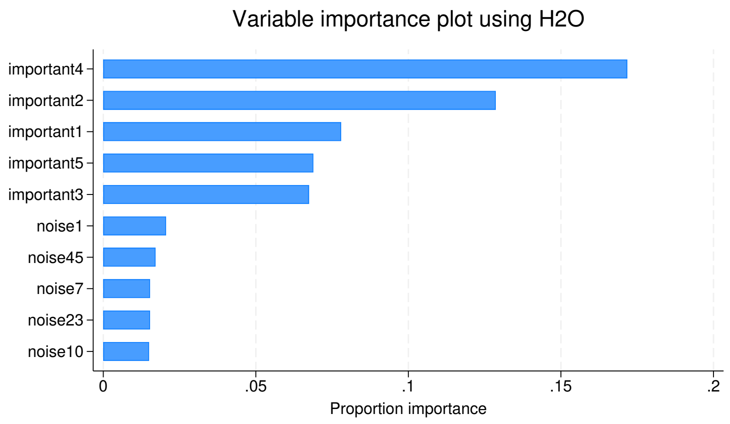


All important predictors are in the top five, but the separation between the important and nuisance predictors is not drastic. We can improve this by tuning the model.

We use a 3-fold modulo cross-validation and 500 trees. For illustration purposes, we train only hyperparameters that control the depth or complexity of the tree, `maxdepth()`, and the number of training samples used to build a tree, `samprate()`. We use the AUC metric for training.

```
. h2oml rfbinclass y important1-important5 noise1-noise45, h2orseed(19)
> cv(3,modulo) ntrees(500) maxdepth(5(1)7) samprate(0.4(0.1)0.6)
> tune(metric(auc))
Progress (%): 0 100
Random forest binary classification using H2O
Response: y
Frame:                                 Number of observations:
  Training: sim                                    Training =  1,000
                                           Cross-validation =  1,000
Cross-validation: Modulo               Number of folds      =      3
Tuning information for hyperparameters
Method: Cartesian
Metric: AUC

─────────────────┬──────────────────────────────────────────────────
                 │                 Grid values
 Hyperparameters │          Minimum         Maximum         Selected
─────────────────┼──────────────────────────────────────────────────
 Max. tree depth │                5               7                6
   Sampling rate │               .4              .6               .5
─────────────────┴──────────────────────────────────────────────────

Model parameters
Number of trees      = 500
              actual = 500
Tree depth:                            Pred. sampling value =     -1
           Input max =   6             Sampling rate        =     .5
                 min =   6             No. of bins cat.     =  1,024
                 avg = 6.0             No. of bins root     =  1,024
                 max =   6             No. of bins cont.    =     20
Min. obs. leaf split =   1             Min. split thresh.   = .00001
Metric summary
──────────────────┬──────────────────────
                  │                Cross-
           Metric │   Training  validation
──────────────────┼──────────────────────
         Log loss │   .6169953    .6233988
 Mean class error │   .3141157     .340729
              AUC │   .7528826    .7385296
            AUCPR │   .7392935    .7251183
 Gini coefficient │   .5057653    .4770591
              MSE │   .2130054    .2160959
             RMSE │   .4615251    .4648612
──────────────────┴──────────────────────
```

From the tuning output, the respective selected best values for `maxdepth()` and `samprate()` are 6 and 0.5. Let's plot the variable importance again.



Now there is a clearer separation between the important and nuisance predictors.

◁

Gradient boosting Poisson regression

▷ Example 14: Explaining Poisson regression predictions

In example 7 of [H2OML] ***h2oml gbm***, we demonstrated how to perform a gradient boosting Poisson regression. In this example, we want to explain the Poisson regression predictions using that model. We repeat some of the steps from that example below and fit the final model.

We start by initializing an H2O cluster, opening the dataset in Stata, and importing the dataset to an H2O frame.

```
. h2o init
  (output omitted)
. use https://www.stata-press.com/data/r19/runshoes
(Running shoes)
. _h2oframe put, into(runshoes)
Progress (%): 0 100
. _h2oframe change runshoes
```

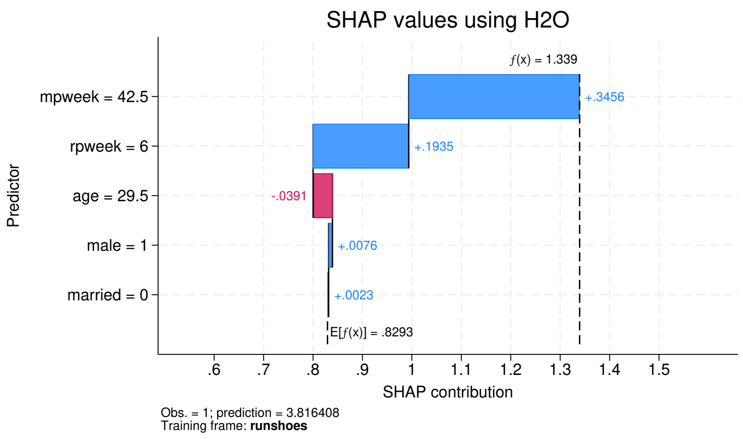
To perform a Poisson regression with h2oml gbregress, we specify the loss(poisson) option.

```
. h2oml gbregress shoes rpweek mpweek male age married trunning, h2orseed(19)
> loss(poisson)
Progress (%): 0 100
Gradient boosting regression using H2O

Response: shoes
Loss:     Poisson
Frame:                                       Number of observations:
  Training: runshoes                                    Training =     60

Model parameters

Number of trees       =  50              Learning rate         =     .1
              actual =  50              Learning rate decay =      1
Tree depth:                              Pred. sampling rate =      1
           Input max =   5              Sampling rate         =      1
                 min =   2              No. of bins cat.      =  1,024
                 avg = 2.9              No. of bins root      =  1,024
                 max =   4              No. of bins cont.     =     20
Min. obs. leaf split =  10              Min. split thresh.    = .00001

Metric summary
```

Metric	Training
Deviance	.3649675
MSE	1.064175
RMSE	1.031589
RMSLE	.2691122
MAE	.7149171
R-squared	.4885824

Next we explain the prediction for the first observation in the runshoes frame by using the h2omlgraph shapvalues command; see [H2OML] **h2omlgraph shapvalues**. You can follow the same steps to explain predictions for other observations.

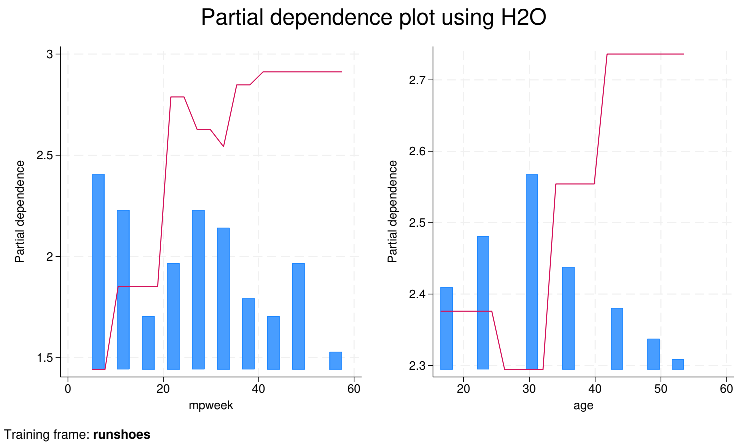
```
. h2omlgraph shapvalues, obs(1) xlabel(0.6(0.1)1.5)
```



The blue bars represent predictors that increase the probability of purchasing running shoes, whereas the red bars represent predictors that decrease it. For this observation, running 42.5 miles per week has a positive effect on the number of shoes purchased, whereas an age of 29.5 has a negative effect.

We continue our analysis and produce a PDP for the predictors `mpweek` and `age` by using the `h2omlgraph pdp` command.

```
. h2omlgraph pdp mpweek age, combineopts(cols(2))
```



The PDP (red line) supports the previous result. Specifically, in the graph for `age` on the right, we observe a noticeable decrease in PDP roughly between ages 25 and 30, which implies a negative effect of age on buying running shoes. But after age 30, the effect is positive.

◁

References

Davis, J., and M. Goadrich. 2006. "The relationship between precision-recall and ROC curves". In *Proceedings of the 23rd International Conference on Machine Learning*, 233–240. New York: Association for Computing Machinery. https://doi.org/10.1145/1143844.1143874.

De Cock, D. 2011. Ames, Iowa: Alternative to the Boston housing data as an end of semester regression project. *Journal of Statistics Education* 19(3). https://doi.org/10.1080/10691898.2011.11889627.

Raschka, S. 2020. Model evaluation, model selection, and algorithm selection in machine learning. arXiv:1811.12808 [cs.LG], https://doi.org/10.48550/arXiv.1811.12808.

Valdenegro-Toro, M., and M. Sabatelli. 2023. "Machine learning students overfit to overfitting". In *Proceedings of the Third Teaching Machine Learning and Artificial Intelligence Workshop*, edited by K. M. Kinnaird, P. Steinbach, and O. Guhr, vol. 207: 46–51. Clearwater Beach, FL: Proceedings of Machine Learning Research.

Wright, M. N., A. Ziegler, and I. R. König. 2016. Do little interactions get lost in dark random forests? *BMC Bioinformatics* 17: art. 145. https://doi.org/10.1186/s12859-016-0995-8.

Also see

[H2OML] **Intro** — Introduction to machine learning and ensemble decision trees

[H2OML] **Glossary**